industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

# UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

# Manejo de un Torno Paralelo Virtual en un entorno inmersivo

**TRABAJO FIN DE GRADO**

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

**Autor:** **Antonio Vicente Martínez Guardiola**

Director: Julián Conesa Pastor

Cartagena, 9 de noviembre de 2020

Universidad Politécnica de Cartagena

# RESUMEN

El proyecto que se presenta a continuación consiste en la realización de una aplicación de realidad virtual para el manejo de un torno en un entorno inmersivo. Para ello, este se basará en un modelo de torno concreto, ubicado en el edifico ELDI de la Universidad Politécnica de Cartagena, de manera que sus alumnos puedan disponer de un soporte digital para el conocimiento de esta máquina herramienta.

Puesto que la máquina data de 1989, no se encontraron planos de los cuales poder sacar las medidas de los componentes. Así pues, la primera etapa del proyecto consistirá en tomar todas las medidas necesarias de los diferentes elementos que presenta la máquina herramienta para reproducir fielmente el modelo.

A partir de los croquis que se tomen del torno se realizará el modelo mediante el software de modelado 3D que nos brinda SolidWorks. Este completo y sencillo programa permitirá reproducir cada uno de los elementos externos que se encuentran en la máquina herramienta y formar un ensamblaje que posteriormente será necesario exportar en otro formato.

El modelo será exportado hasta otro software con soporte en realidad virtual: Unity. Este programa no se utiliza exclusivamente para VR, se trata de un motor de videojuegos que habrá que adaptar e integrar una serie de configuraciones para su correcto funcionamiento. De esta manera se podrá programar el funcionamiento del torno para la realización de operaciones básicas de cilindrado exterior o refrentado.

**Palabras clave**: Realidad Virtual (VR); Torno; Modelado 3D; Programación en C#.

# ABSTRACT

The project presented below consists of the development of a Virtual Reality application for the operation of a lathe in an immersive environment. This project is based on a specific lathe model, which is located in the ELDI Laboratories in Universidad Politécnica de Cartagena, so that students will be able to have a digital support for the knowledge of this machine tool.

Given that the said machine tool dates back to 1989, it was not possible to find any plans in order to extract the dimensions of the components. Thus, the first phase of the project consists of the necessary measuring of the different parts of the lathe, so as to faithfully reproduce the model.

On the basis of the machine tool sketches, the model has been created by using SolidWorks 3D modelling software. This complete and simple program allows us to reproduce each external component which can be found on the machine tool and to create an assembly that will be necessarily exported in another 3D format at a later time.

The model has been exported to another Virtual Reality software: Unity. This program is not only used for VR, but it is a game engine which needs to be adapted, and it is also necessary to integrate several settings for its proper operation. Therefore, the lathe operation can be programmed, and it is possible to perform basic external turning and facing operations.

# AGRADECIMIENTOS

Este proyecto se ha conseguido desde la ilusión de poder enorgullecer a todas las personas que me acompañan en mi vida, así como una forma de superación personal. En primer lugar, me gustaría dar las gracias a mis padres por educarme en la cultura del esfuerzo, la humildad y la generosidad, y, en definitiva, por haber sido, junto con mi hermana, mi principal sostén durante toda esta etapa que hoy termina.

En general, me gustaría dedicar este trabajo a toda mi familia, y en especial a mis abuelos, que tanto me han cuidado a lo largo de los años y de los que tanto he aprendido y sigo haciéndolo.

A su vez, quiero dar la gracias a mi director y el que fuera mi profesor de Diseño Industrial, Julián F. Conesa Pastor. Gracias por introducirme en este nuevo "mundo virtual" y por guiarme a lo largo de todo el transcurso del proyecto.

También quería dar las gracias a Juan Carlos Sánchez Aarnoutse, por su disposición a la hora de prestarnos el soporte físico necesario para poder llevar a cabo mi proyecto, sin el cual no habría sido posible. Y a Pedro Belmonte Alfaro, por su gran amabilidad y disponibilidad para abrirnos el laboratorio del ELDI y asesorarme en el funcionamiento del torno.

Finalmente, quiero dar las gracias a todos aquellos compañeros con los que he luchado mano a mano durante toda la carrera y de los que me siento muy orgulloso de tenerlos también como amigos para el resto de los años.

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# ÍNDICE DE SCRIPTS

# ÍNDICE DE TABLAS

# CAPÍTULO 1. INTRODUCCIÓN

## 1.1. Motivaciones

Hasta hace escasos años, la realidad virtual se veía ligada únicamente al mundo del entretenimiento. Sin embargo, gracias al desarrollo tecnológico y a la digitalización Industrial, se han visto incrementadas las diferentes soluciones en donde estas tecnologías cobran mayor sentido.

La capacidad de poder ofrecer espacios virtuales en donde introducir diseños o modelos en 3D para simular o recrear situaciones por medio de la programación en realidad virtual, así como el avance del tejido empresarial hacia la Industria 4.0, han sido factores determinantes en la realización de este proyecto.

## 1.2. Objetivos

El objetivo principal de este proyecto será simular el manejo de un torno virtual en un entorno de inmersión recreado por un ambiente industrial en el que el usuario pueda desplazarse e interactuar directamente con la máquina. De esta manera, el alumno podrá disponer de soporte digital para el conocimiento de esta máquina y, si dispone de un equipo de VR, manejar dicho torno en operaciones de torneado recto con la posibilidad de la no presencialidad.

Este trabajo pretende ser la punta de lanza de otro ambicioso proyecto que dirige el director de este TFG, por medio del cual se quiere reproducir el laboratorio de fabricación de la Universidad Politécnica de Cartagena.

# CAPÍTULO 2. ESTADO DEL ARTE

## 2.1 Introducción a la Realidad Virtual

"Representación de escenas o imágenes de objetos producida por un sistema informático, que da la sensación de su existencia real" [1].

"La Realidad Virtual es una simulación interactiva por computador desde el punto de vista del participante, en la cual se sustituye o se aumenta la información sensorial que recibe" (A. Rowell, 2009).

William y Alan [2] introducen los conceptos de *Virtual World*, *Inmersión*, *Sensory Feedback* e *Interactivity* como elementos fundamentales dentro de cualquier sistema de realidad virtual. Estos se resumirán a continuación en tres, según explica la asignatura de realidad virtual de la Universidad Politécnica de Cataluña [3]. Estos son:

- Simulación Interactiva
- Interacción Implícita
- Inmersión Sensorial

### Simulación Interactiva

Se habla de una simulación puesto que se recrea una escena virtual en la que se puede realizar una serie de acciones que en realidad no se están llevando a cabo. La definición de simulación se encuentra dentro de una gran variedad de campos de investigación como la química, la física, la Industria o la sociedad en sí misma, con el fin de realizar estudios para evaluar el comportamiento de ciertos parámetros y variables.

Por otro lado, se dice que es interactiva ya que el usuario no se encuentra como un mero espectador. El usuario se involucra activamente en ese mundo virtual de manera que puede interactuar con él, alterando en tiempo real y de forma directa, a las imágenes que se proyectan en la escena.

### Interacción Implícita

En contraposición a la interacción implícita se encuentra lo que se conoce como Interacción clásica. La diferencia más significativa que se puede encontrar entre ambas consiste en que el usuario llega a olvidar los dispositivos que llevan a éste a ese nuevo mundo virtual, de manera que pasa a formar parte activa de ella cuando se habla de interacción implícita.

La interacción clásica consiste en una comunicación explícita entre el usuario y el sistema. Esto es, el usuario debe expresar al sistema la voluntad de realizar cualquier acción utilizando el esquema de comunicación predefinido por la propia aplicación. Esto supone que el usuario deba conocer cuáles son los comandos que están asociados a realizar dicha acción (los dispositivos tradicionales de interacción clásica son tanto el teclado como el ratón).

Sin embargo, cuando nos encontrados sumergidos en un sistema de realidad virtual, es la propia aplicación la que captura la voluntad del usuario. Esto se consigue mediante los diferentes sensores de los que se hablará más adelante y que permiten capturar nuestros movimientos naturales. Un ejemplo básico que se puede mencionar se encuentra en el rastreo de los movimientos de la cabeza o nuestras propias manos. En este caso el usuario

no debe transmitir al sistema la acción que va a realizar, sino que este simplemente la lee a través de nuestros movimientos naturales.

**Inmersión Sensorial**

"La realidad virtual implica presentar a nuestros sentidos un entorno virtual generado por un ordenador" [4]. Esto implica la desconexión de los sentidos dentro del mundo real, para ser conectados en el mundo virtual.

Desde joven, el ser humano estudia los cinco sentidos que le permiten sentir y estar conectado con el mundo que le rodea, estos son: la vista, el olfato, el oído, el gusto y el tacto. Sin embargo, existen otros sentidos a parte de estos básicos, como puede ser el sentido del equilibrio, fundamental en las aplicaciones de realidad virtual donde es posible sufrir mareos. De esta manera, todas estas entradas sensoriales extra, aseguran a nuestra mente un flujo de información mediante la estimulación de nuestro sistema nervioso, permitiendo una experiencia más realista.

Dentro de nuestros sentidos básicos, nuestros ojos perciben la mayor parte de la información que recibimos de nuestro entorno. Cuando miramos una imagen en la pantalla de nuestro ordenador, por mucha resolución que esta pueda tener, nuestra mente no percibe que esta tenga una existencia material, simplemente vemos una imagen bidimensional (aunque esta imagen sea un cubo) proyectada dentro de la superficie de nuestra pantalla. Esto no ocurre así cuando se observa  a través de los visores de realidad virtual, donde los objetos cobran profundidad y "flotan" en la escena dentro del espacio que envuelve al espectador. Esto se consigue llevar a cabo a través de la visión estereoscópica.

Visión estereoscópica

"La visión estereoscópica es un proceso que se realiza de manera natural cuando un observador mira simultáneamente dos imágenes de un mismo objeto, que han sido captadas desde dos posiciones distintas. Cada ojo ve una imagen y el resultado de ese proceso es la percepción de profundidad o tercera dimensión". "La percepción de profundidad se puede lograr involucrando las sombras, el tamaño relativo de los objetos o mediante la perspectiva" [5].

Esto es, la visión estereoscópica se basa en proporcionar dos imágenes (una por cada ojo) ligeramente distintas, de manera que nuestro sistema visual deduce la profundidad de estos objetos a través de estas diferencias.

"The sound and music are 50% of the entertainment" (George Lucas). El oído es otro factor sensorial importante a la hora de "sentir" la realidad virtual. Es por ello por lo que también se encuentran dispositivos centrados en mejorar la experiencia del usuario a través del sistema auditivo. Un ejemplo de ello son los audífonos Entrim 4D de Samsung. Esos audífonos permiten engañar al oído interno para regular el equilibrio y el movimiento dentro del oído logrando desarrollar 30 diferentes patrones de movimiento. Esto permite reducir en gran medida algunas sensaciones derivadas de las aplicaciones de realidad virtual como el mareo y las náuseas de manera que ahora nuestro cuerpo será más consciente de lo que está viendo.

## 2.2 Historia y antecedentes

### 1836-Estereoscópico de Charles Wheatstone

Se puede atribuir el origen de la realidad virtual a Charles Wheatstone, cuando en el año 1836 inventó el estereoscópico. El invento monta, sobre una estructura de madera, un soporte para las imágenes que quieren visualizarse, y cuatro espejos a través de los cuales conseguimos ver las imágenes mediante el efecto estereoscópico explicado anteriormente. Además, introduce un eje vertical para cambiar el tamaño de estas.

*Figura 1. Estereoscópico de Wheatstone [6]*

En un principio se divulgó como un invento científico, sin embargo, sus aplicaciones se redujeron a los de la juguetería óptica hasta que con la aparición de la fotografía se convertirá en un fenómeno visual. De esta manera, no sería esto, sino la invención del puente de Wheatstone utilizado para medir resistencias eléctricas lo que le daría mayor reconocimiento profesional.

### 1929-Blue Box

Un siglo después, en el año 1929, se perseguía crear un simulador para que los pilotos de guerra estadounidenses pudieran entrenarse en las mismas condiciones de vuelo real. Este fue el objetivo de Edin A. Link con su invento: el Link Trainer, también conocido como Blue Box.

*Figura 2. Blue-Box de Edwin A. Link [7]*

Consiguió que el aparato se pudiera desplazar siguiendo las órdenes del piloto, el cual se comunicaba por radio para un mayor realismo, simulando al mismo tiempo que este se movía y tenía la misma velocidad de reacción que un avión de la época ante condiciones meteorológicas adversas.

Con la llegada de la Segunda Guerra Mundial se habían fabricado más de diez mil unidades, que siguieron distribuyéndose hasta pasada la guerra. Desde entonces estos simuladores no han parado su desarrollo. El verdadero avance llegaría con la introducción de los ordenadores en 1972

**1957-Sensorama**

En el año 1957, gracias al impulso "futurista" que había en el cine de la época, el director Morton Heilig inventó lo que podría haber sido la primera máquina de inmersión sensorial. Una cabina que permitía estimular la vista, el olfato, el oído y el tacto todo ello mezclado con imágenes 3D estereoscópicas y sonido estéreo real.


*Figura 3. Sensorama [8]*

Este nuevo invento simulaba un paseo en bicicleta en el que el usuario podía sentir la naturaleza a través de olores y sonidos, además de introducir vibraciones que le daban la sensación de ir desplazándose. Sin embargo, resultó ser un proyecto bastante caro debido a la captura de imágenes realizada a partir de tres cámaras de 35mm por lo que no llegó a comercializarse.

**1968-La Espada de Damocles**

Posteriormente en el año 1968 Ivan Sutherland materializó lo que sería el primer visor de realidad virtual, un proyecto que describió como "La espada de Damocles". Este invento permitiría al usuario modificar las imágenes que visualizaba en escena, en función del desplazamiento y la orientación de la cabeza del propio usuario. Tenía un peso considerable de manera que a partir de un brazo mecánico se sujetaban la mayoría de los componentes formados por: un multiplicador de matriz, un sistema de audio, un generador de vectores, un ordenador, un divisor de corte y un sensor para la posición de la cabeza

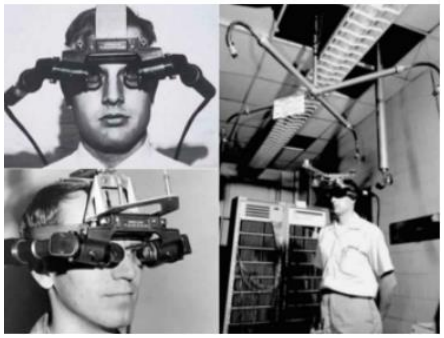


*Figura 4. Espada de Damocles [9]*

En aquella época los avances eran aun pobres, de manera que el visor solo reproducía polígonos y su visualización era bastante pobre. Aun así, esto sentaría las bases de la realidad virtual que hoy conocemos a partir del uso de dispositivos HMD y Posicionadores de los que se hablará más adelante. Además, dado que el casco permitía visualizar parte del entorno real, debido al mal asilamiento, se introduciría por primera vez el concepto de realidad aumentad.

**1970-BOOM**

Dos años después, la empresa Fake Space Systems Inc presenta BOOM (Binocular Onmi Orientation Monitor), un sistema formado por una pantalla estereoscópica y un sistema de auriculares dentro de una carcasa en la cual el usuario introducía la cabeza. Este sistema iba sujeto mecánicamente mediante brazos articulados y el seguimiento de la cabeza se realizaba a partir de los eslabones del brazo que sujetaba la carcasa.

**1977- Sayre Glove**

Para manipular objetos del entorno virtual, Richard Sayre inventa el primer guante sensitivo a la flexión al que denomina: Sayre Glove. Para poder simular el movimiento de los dedos en la escena virtual, estos guantes, con unos tubos de fibra óptica, se iluminaban por un extremo y en función de la intensidad de luz recibida se podía calcular el grado de apertura de la mano.

*Figura 5. Sayre Glove. Fuente: VPL DataGlove*

Este no sería el único guante que se inventaría en aquellos años, ya que seguirían su desarrollo durante los años ochenta con el guante de Bell Labs de AT&T del Dr. Gary Grimes o los guantes creados en colaboración por Thomas Zimmerman y Jaron Lanier con ventas tan exitosas como a la NASA.

**1980-Super Cockpit**

En el año 1980, Thomas Furness [10] desarrolló un simulador de vuelo llamado Super Cockpit. Este proyecto fue utilizado por pilotos para poder realizar entrenamientos. La cabina proyectaba mapas tridimensionales, imágenes infrarrojas y de radar en tiempo real. El piloto controlaba el avión utilizando gestos, palabras y movimientos oculares.

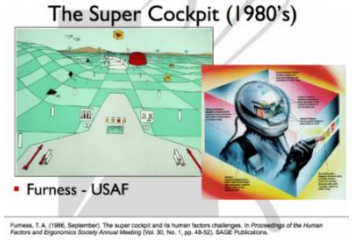



*Figura 6. Super Cockpit de Thomas Furness [10]*

**1986-VIVED**

No sería hasta el año 1986 cuando la NASA daría a conocer lo que hoy se conocen como visores de realidad virtual: VIVED (virtual visual environment display). Lo haría en el año 1986 en la feria electrónica del consumo: CES

*Figura 7.VIVED (for virtual visual environment display) [11]*

Estas gafas tenían un campo de visión de 120º, gracias a la utilización de dos pantallas LCD. Incorporaba control por voz y un sistema de reconocimiento de gestos mediante unos guantes. Para la orientación del individuo se proponía un traje repleto de sensores que pudieran captar los movimientos del usuario.

**1991-SEGA VR y BOY de Nintendo**

En el año 1991 llegaría la realidad virtual a un público más convencional a través de la industria del videojuego. SEGA VR y virtual BOY de Nintendo.

*Figura 8.Nintendo BOY y SEGA VR*

No obstante, la primera de ellas nunca llegó al mercado. Según argumentó SEGA, “El resultado del dispositivo era tan real que temían por el bienestar de sus clientes”. Por otro lado, la consola que si sacó Nintendo al mercado resultó ser un fracaso. La peor consola vendida hasta la fecha por esta marca.

**2012-Palmer Luckey**

No sería hasta 2012 cuando la empresa de Palmer Luckey se daría a conocer con un dispositivo que ofrecía un ángulo de visión de 90º en horizontal y una vertical de 110º en una perspectiva 3D estereoscópica: las Oculus Rift. Tanto fue su éxito que Facebook compró la empresa en 2014 comenzando una carrera dentro del mundo de la Realidad Virtual.

Con estas gafas la realidad virtual volvería a ponerse de moda. Esto, junto a los avances tecnológicos más recientes, han permitido dotar a la realidad virtual del atractivo que esta merece para aplicaciones de sectores estratégicos muy distintos.

## 2.3 Desarrolladores actuales de Realidad Virtual

Dentro del amplio mercado de realidad virtual, podemos destacar los siguientes desarrolladores de software y hardware para esta tecnología incipiente:

### Oculus VR

Esta es una de las compañías pioneras en el desarrollo de productos de software y hardware para realidad virtual. Fue fundada por Palmer Luckey, Brendan Iribe, Michael Antonov, Jack McCauley y Nate en julio de 2012. Desde marzo de 2014 pertenece a Facebook, que adquirió Oculus VR por un valor de 2.3 millones de dólares.

En cuanto a los productos lanzados por la compañía encontramos:

- Oculus Rift (2016)
- Oculus Go (2017)
- Oculus Quest (2018)
- Oculus Rift S (2019)

También cabe mencionar que, en 2015, junto con Samsung, Oculus desarrolló Samsung Gear VR para los dispositivos Samsung Galaxy.

### SONY

Conocido en un primer momento como “Project Morpheus, SONY desarrolló un visor de realidad virtual para funcionar con su plataforma más vendida, la Playstation 4, este proyecto se acabó llamando: PlayStation VR. Destinada a la industria del entretenimiento.

PlayStation VR se mostraría por primera vez al público en “The Tonight Show Starring Jimmy Fallon” y como concepto durante el evento de E32014.

### HTC/Valve

De la mano de HTC y Valve nacen las gafas de realidad virtual VIVE. Estas gafas se presentaron al mundo durante el World Mobile Congress en marzo de 2015 así como en el CES de 2016, ganando más de 20 premios.

Dentro de esta compañía encontramos dos series de productos:

- Cosmos series
- Pro-series

En la línea de Valve Corporation también cabe mencionar a la plataforma de distribución digital de videojuegos: Steam. Esta plataforma lanzaría en 2014 una nueva iniciativa para adaptar la realidad virtual a los juegos digitales, Steam VR.

### Microsoft

Una de las grandes novedades que ha introducido Microsoft en las últimas actualizaciones de Windows 10 lanzada en 2017 es la “Windows Mixed Reality”. Esta es una mezcla entre la realidad virtual y la realidad aumentada. Para acceder a ella Microsoft ha creado el portal de realidad mixta dentro de Windows 10 en el que el usuario puede navegar por el escritorio de manera virtual. Esta herramienta será compatible para la mayoría de los cascos de inmersión que podemos encontrar en la actualidad, aun así, Microsoft también presenta su propio dispositivo: Microsoft HoloLens. Estos es un casco equipado con unas gafas inteligentes inalámbrica e independiente con Windows 10 Holographic.

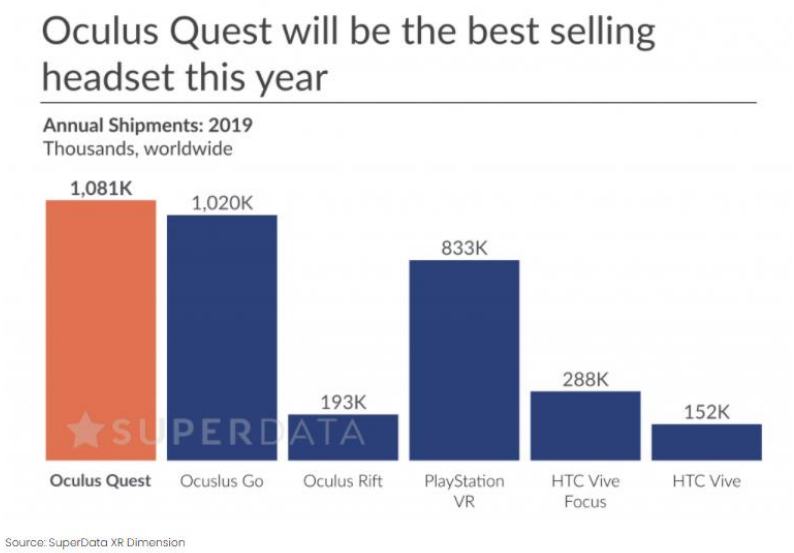



*Figura 9. Mercado de los principales desarrolladores de VR [12]*

En la figura 9 podemos ver el tamaño de mercado de los principales equipos de realidad virtual por ventas durante el año 2019.

### 2.4 Diferencias entre la Realidad Virtual (VR) y la Realidad Aumentada (AR)

A parte de la realidad virtual, se ha hablado de la realidad aumentada, incluso de la realidad mixta, pero ¿qué relación existe entre todas ellas? En el año 1994 Paul Milgram y Fumio Kishinoel [13] idearon el concepto de "Virtuality continuum" es decir, continuo de la virtualidad, por el cual se podía explicar la relación entre estas realidades. Este concepto se ve clarificado con la siguiente ilustración:


*Figura 10. Continuo de la virtualidad [13]*

Si nos desplazamos hacia la derecha de la recta, los estímulos que reciben nuestros sentidos son de generación virtual en su totalidad, mientras que, si nos movemos en sentido opuesto, tendremos el mundo real, el que ven nuestros ojos de manera natural.

La realidad mixta se sitúa en la mitad de estos dos extremos, digamos que es una mezcla entre el mundo virtual y lo real, una realidad donde se añaden elementos artificiales que solo existen en el plano del "metaverso".

Ahora bien, si nos desplazamos un poco hacia la izquierda, hacia el mundo real, tendríamos la realidad aumentada, esto es, la combinación de elementos virtuales dentro

del espacio real. En este caso todo lo que observamos está sobre un entorno real y no es preciso utilizar ningún visor.

Por otro lado, si nos desplazamos hacia la derecha, hacia el mundo virtual, nos encontraríamos con la virtualidad aumentada, esto es, la combinación de objetos reales en el mundo virtual.

De manera sencilla, se dice que la realidad mixta abarca estas dos mezclas de realidades entre lo real y lo virtual.

Finalmente, la realidad extendida (XR) abarca todas las realidades posibles. En lugar de decir Realidad Virtual (RV) o Realidad Aumentada (RA) basta con cambiar la última letra por una X, para decir RX.

## 2.5 Aplicaciones de la Realidad Virtual

El mundo del entretenimiento y la industria de los videojuegos ha sido el campo de desarrollo por excelencia de la realidad virtual, sin embargo, vemos como cada vez más, esta tecnología empieza a cobrar sentido en otros campos como la medicina, la cultura, la arquitectura, la Industria o la educación. La figura 11 muestra las predicciones realizadas sobre la evolución del mercado para el año 2025 en billones de dólares, en función de las diferentes aplicaciones.

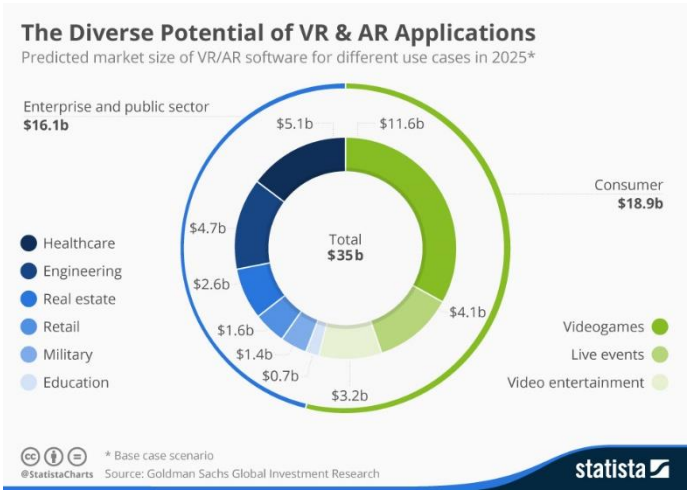



*Figura 11. Predicción para 2025 sobre el tamaño del mercado de VR según sus aplicaciones [14]*

### Medicina

Además de funcionar como tratamiento, la realidad virtual en medicina permite mejorar la formación mediante la simulación o la representación de la anatomía de un paciente en tres dimensiones.

"El valor más importante de la simulación como herramienta educativa consiste en que con los elementos adecuados; espacios (consultorios, habitaciones, quirófanos, salas de trauma, unidades de cuidados intensivos, salas de parto y quirófanos) construidos a escala real y dotados de elementos virtuales, se pueden generar los escenarios, situaciones cotidianas y complejas donde el docente y el estudiante pueden repetir, corregir y perfeccionar su acto médico…" [15].

Dentro del CSIC también encontramos grupos de investigación en los que trabajan por la inclusión de esta tecnología para la planificación y la optimización de procesos quirúrgicos.

**Militar**

Algunos ejércitos utilizan la realidad virtual también para el entrenamiento de los militares en entornos de combate simulados. El ejército de los EEUU acude a la realidad virtual para realizar simuladores de campos de batalla para conocer las condiciones del mismo, así como simuladores de vuelo u otros vehículos pesados, lo que permite minimizar los riesgos y mitigar el coste que supone movilizar vehículos de esa magnitud [16].

**Arquitectura**

Mediante la realidad virtual los arquitectos presentan mayores ventajas a la hora de visualizar el espacio e interactuar con los diferentes proyectos incluso antes de ser construidos. También se puede utilizar como forma de presentación de proyectos a clientes para evitar desplazamientos físicos si así se requiere.

**Educación**

Dentro del ámbito de la educación, la realidad virtual permite a los alumnos retener mejor los conocimientos adquiridos en las aulas, además de ayudar a aquellos estudiantes con mayores problemas de aprendizaje.

"…la incorporación de la Realidad Virtual supondrá un salto cualitativo muy importante en el aprendizaje de disciplinas o áreas de conocimiento, especialmente en aquellas en las que resulta difícil visualizar los procesos estudiados. La utilización de modelos virtuales permite obtener un sentido del espacio 3D del que carece cualquier otro sistema de representación gráfica. Además, se trata de una tecnología bastante intuitiva en cuanto a su uso y que consigue facilitar la explicación de conceptos complejos o abstractos" [17].

**Industria**

Fue la industria química una de las primeras en implementar esta tecnología en las principales compañías de Petróleo y Gas para su uso en servicios de aplicación en campo, operaciones de simulación, y laboratorio. En este tipo de compañías se cuenta con instalaciones de gran tamaño y de mayor complejidad, operando las 24h, donde los trabajadores están expuestos a situaciones de emergencia para las cuales deben ser entrenados.

"Mediante el estudio de los procedimientos en este mundo virtual, ingenieros, geólogos, planificadores, expertos en seguridad y trabajadores pueden identificar problemas, explorar las distintas opciones y determinar la mejor solución, sin interrumpir las operaciones, evitando costosos errores, daños a los equipos y riesgos de seguridad o medioambientales" [18].

La realidad virtual y la realidad Aumentada son dos tecnologías imprescindibles para la transformación digital de las empresas hacia la Industria 4.0. En esta nueva revolución industrial, la tecnología se pone al servicio de este sector para mejorar aspectos como la productividad, la seguridad y en general la optimización de los procesos industriales. Lo que se pretende es tener empresas inteligentes conectadas a tiempo real para tener acceso a los distintos elementos del sistema, y es por ello que la realidad virtual y la realidad aumentada son imprescindibles para llevar este proceso a cabo.

El uso de la realidad virtual o aumentada dentro de la industria puede aprovechar una gama amplia de fuentes de datos para resolver tareas complejas, mejorar la formación de nuevos trabajadores para realizar trabajos cualificados o complejos, facilitar la formación cruzada, acceso de especialistas de forma remota, creación de salas de control virtuales para el control de planta, etc.

Existen algunos sectores que invierten grandes cantidades de recursos para realizar prototipos. Hablamos de industrias como la automovilística donde se realizan construcciones, de coste inferior, para analizar las características y su uso real. Estos costes pueden ahorrarse mediante la implementación de realidad virtual, que permite su simulación. BMW ha sido la primera en implantar esto para visualizar diferentes acabados en sus vehículos, así como otras características físicas [19].

Mediante la realidad virtual se pueden simular entornos dentro de los cuales los operarios puedan interactuar para realizar operaciones de mantenimiento y resolución de averías. De esta manera el operario solo tendría que ponerse un equipo de visualización y observar el funcionamiento de la máquina, así como realizar las acciones necesarias para solucionar la incidencia.

# CAPÍTULO 3. ARQUITECTURA DE UN SISTEMA DE REALIDAD VIRTUAL

## 3.1 Elementos de Hardware

### 3.1.1 Periféricos de entrada

También denominados sensores, estos son aquellos encargados de capturar las acciones del usuario y transmitir dicha información hasta el sistema. Entre los elementos más básicos se encuentran: Posicionadores, guantes de datos, registros de voz y los dispositivos de entrada 3D.

- **Posicionadores**

Son los encargados de recoger toda la información relacionada con la posición y la orientación del participante. En realidad virtual, la orientación de la cabeza del usuario se mide sobre los ejes de rotación X, Y y Z:

**Yaw (azimuth)**. Mide el Angulo de rotación respecto al eje vertical.

**Pitch (elevation)**. Mide el ángulo de rotación respecto del eje horizontal.

**Roll**. Mide el ángulo de rotación respecto al eje determinado por la dirección de nuestra mirada.

Para el seguimiento de la cabeza se utilizan acelerómetros, giroscopios y magnetómetros que se incorporan dentro los efectores visuales, cuyas características veremos más adelante. Estos sistemas permiten actualizar la imagen cuando el usuario mueve la cabeza.

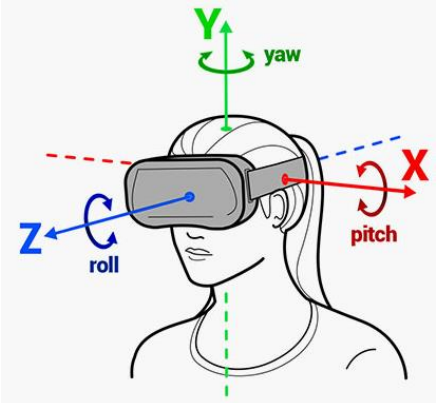



*Figura 12. Rotación de los ejes principales [20]*

La mayoría de los desarrolladores de este tipo de efectores visuales permiten posicionar al usuario en la escena mediante la utilización de leds infrarrojos que se colocan sobre el casco. También se suelen colocar de manera estratégica sobre los dispositivos de entrada 3D como es el caso del sistema Oculus Rift para realizar el rastreo de movimientos, no solo de la cabeza. Este sistema es conocido como Oculus Constellation.

*Figura 13. Oculus Constellation [21]*

Otro sistema que encontramos lo incorpora la marca HTC Vive, denominado Lighthouse. En este caso se disponen de dos dispositivos que emiten láseres, uno en dirección vertical y otro en la horizontal, que van a ser recibidos por los dispositivos de entrada o mandos que incorpora este equipo. Esto hace que se iluminen una serie de leds de manera que, al

realizar el barrido de la zona de juego, se detecten los sensores que han sido alcanzados, así como el tiempo que ha pasado desde que se produjo el flash de los leds.

*Figura 14 HTC Vives estaciones.*
*Fuente:HTC.com*

- **Guantes de datos**

Los guantes de datos se empezaron a utilizar en realidad virtual para registrar la posición de la mano dentro de la escena y para poder interactuar con objetos mediante gestos naturales. La posición de los dedos es detectada mediante los ángulos de deflexión de la mano.

Esta solución ha dejado de tener interés entre la mayoría de las aplicaciones destinadas al público convencional y se ha dado paso a la utilización de los llamados controladores que permiten posicionar la mano en la escena gracias a la lectura de los Posicionadores y además permiten interactuar y moverse a través del mundo virtual. Los denominados dispositivos de entrada 3D.

- **Dispositivos de entrada 3D**

Como se ha dicho ya, estos permiten interactuar de forma más cómoda en el espacio virtual. Entre los más utilizados en la actualidad encontramos los controladores. Estos dispositivos, como se ha mencionado mencionado anteriormente, permiten posicionar al usuario, así como interactuar con los elementos que ofrece el mundo virtual según la mayoría de equipos que encontramos hoy en día. A parte de funcionar como elemento de entrada, algunos controladores presentan efectores hápticos con diferentes modos de vibración que permiten despertar nuestros estímulos sensibles al tacto.

*Figura 15 Controladores HTC [21]*

### 3.1.2 Periféricos de salida

También denominados efectores, permiten transmitir la información generada por el sistema para estimular nuestros sentidos en forma de imágenes, audio, vibraciones, etc.

Estos se clasificarán en: efectores visuales, efectores auditivos, efectores táctiles y efectores de equilibrio.

#### *3.1.2.1 Efectores visuales*

Tienen la función de estimular el sistema de visión del usuario reproduciendo imágenes con sensación de profundidad. Dentro de estos efectores se verán los siguientes aspectos:

- **Factores de calidad de la visualización**

Algunos de los factores que se deben tener en cuenta son los siguientes:

**Profundidad de color**. Se refiere a la cantidad de bits de información necesarios para representar el color de un píxel (*bpp*). Cuanto mayor sea la profundidad de color se obtendrá una gama más amplia de distintos colores.

**Frecuencia de refresco**. La frecuencia o tasa de refresco hace referencia a las veces que se actualiza una imagen por unidad de tiempo. Este es otro factor fundamental en sistemas de realidad virtual donde se necesitan obtener movimientos fluidos. Se debe distinguir entre la frecuencia de refresco del dispositivo de visualización (*vertical frequency*) y la frecuencia de refresco de la tarjeta gráfica (*frame rate*).

La frecuencia de refresco del dispositivo de visualización hace referencia al número de imágenes diferentes que es capaz de presentar en una unidad de segundo (se expresa en Hz). Por otro lado, la frecuencia de refresco hace referencia a la velocidad con la que se muestran los fotogramas en pantallas. De esta manera, si se trabaja a 60 fotogramas por segundo (*fps*), estamos diciendo que en un segundo se muestran 60 imágenes diferentes.

**Resolución**. Esto es, el número de pixeles que presenta la imagen. Es un aspecto bastante importante en cualquier pantalla donde se proyecte una imagen, pero lo es más en los dispositivos de VR donde la pantalla se encuentra muy cerca de los ojos.

Una resolución muy elevada tampoco es garantía de buena calidad ya que también se debe controlar las dimensiones de la pantalla donde se va a proyectar la imagen. Es por ello por lo que se suele utilizar la resolución angular (número de pixeles para cada grado de visión del usuario).

**Campo visual (field of view, FOV)**. Esta muestra el intervalo del campo visual del usuario que cubre la imagen de la pantalla. Cuanto más se acerque el campo visual del dispositivo al del campo visual humano (aproximadamente de 180 grados en horizontal, y un poco menos en vertical) se percibirá un mayor realismo.

- **Dispositivos de visualización**

**Cascos estereoscópicos**

Dispositivo que se introduce sobre la cabeza del usuario para poder visualizar de forma personal el mundo virtual en tres dimensiones. Se pueden encontrar diferentes tipos de cascos según la clasificación que hagamos. La mayoría cuenta con sensores que permiten obtener la posición y la orientación dentro del espacio de rastreo.

Clasificación:

— **HMD (head mounted display)**. Estos cascos de inmersión sensorial permiten el enfoque mediante una pantalla alineada con cada ojo. Las pantallas de estos dispositivos suelen equipar tecnología LCD, aunque cada vez es más frecuente encontrar dispositivos con pantallas OLED.

*Figura 16. Dispositivo HMD de Lenovo.*
*Fuente: Lenovo.com*

Dentro de los visores HMD se pueden encontrar: Visores de escritorio (que requieren de un equipo externo (PC)), independientes (todo el sistema se encuentra dentro del visor) y finalmente los móviles/carcasas (requieren de un Smartphone conectado al propio visor).

También se pueden diferenciar estos visores en función de los grados de libertad que disponga. Es por ello que existen visores con tres grados de libertad, donde quedarían limitados los desplazamientos en los tres ejes y solo se permite el giro, y los visores con seis grados de libertad que permiten el desplazamiento y el giro en cada uno de sus ejes.

— Los **HCD (Head-Coupled Displays)** equipan pantallas clásicas con cañones de electrones (Cathode Ray Tube, CRT) de alta resolución, pero muy pesadas, de manera que deben ir montadas sobre un soporte mecánico. Este contiene, a su vez, potenciómetros que permiten monitorizar el movimiento del participante.

**Sistemas basados en proyección**

En este caso las imágenes se visualizan en una o varias zonas de proyección. Ahora ya no es una única persona la que puede verse inmersa en la escena de realidad virtual. Las configuraciones más conocidas son: la habitación estereoscópica (CAVE) y la mesa estereoscópica WorkBench).

La habitación estereoscópica consiste en un sistema de visualización formado por una habitación cúbica de 3x3x3 metros, de manera que las imágenes se puedan proyectar por las 6 paredes de esta. Se suele utilizar retroproyección para evitar sombras y proyección directa para el suelo.

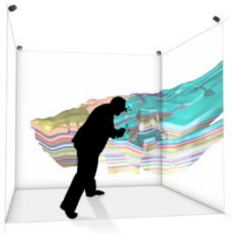


*Figura 17. Modelo CAVE.*
*Fuente: innovaTecno.com*

La mesa estereoscópica consiste en una pantalla en forma de mesa en la cual se proyectan las imágenes mediante dos proyectores de características convencionales. Para visualizar imágenes en 3D se recurren a filtros de doble polarización circular.

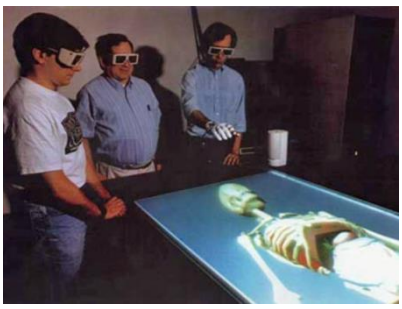


*Figura 18. Mesa Estereoscópica.*
*Fuente: innovaTecno.com*

En las aplicaciones de realidad virtual que utilizan cascos estereoscópicos, el grado de coordinación entre la visión y la mano no es tan alto (requiere una representación virtual de cada mano del usuario) de manera que en aplicaciones precisas como pueden darse en el campo de la medicina, estas pueden resultar muy útiles. Su uso también es recurrido en pantallas de cine 3D.

#### *3.1.2.2 Efectores auditivos*

En el caso de los cascos estereoscópicos suelen equipar, a parte del sistema de visualización, un sistema de audio que permite mejorar la sensación del usuario. Frecuentemente se utiliza el sonido espacial con el que se obtiene un resultado de mayor inmersión en la escena ya que el participante puede ubicar el foco emisor en una posición concreta de esta.

De manera bastante frecuente se recurre al sistema auditivo para crear sensaciones hápticas sin hacer uso de efectores hápticos, que por lo general suelen ser más caros y complejos.

#### *3.1.2.3 Efectores táctiles*

Mediante estos dispositivos se pretende estimular el tacto. Para ello se suelen utilizar guantes táctiles que permite percibir el contacto con algunos objetos virtuales. Esto se consigue mediante descargas eléctricas, también se utilizan sistemas de vibración o incluso cámaras neumáticas para simular la percepción de agarre.

También se pueden encontrar dispositivos de realimentación de fuerza que ofrecen resistencia a hacer movimientos con la mano cuando recibe el impacto virtual de un objeto.

### 3.1.3 Computador

Acabando con los dispositivos de hardware, el ordenador será el encargado de realizar la simulación de forma interactiva, a partir del modelo geométrico en 3D y del software de recogida de datos, simulación física y simulación sensorial.

En caso de trabajar con visores de escritorio, los ordenadores utilizados para soportar sistemas de realidad virtual deben contar con prestaciones gráficas avanzadas para

cumplir con el rendimiento óptimo de la aplicación. Por ello, para este tipo de tecnología se utilizan ordenadores tipo "Gaming" que cuentan con las especificaciones necesarias o incluso existen ordenadores optimizados especialmente para VR.

| | |
|---|---|
| **TARJETA GRÁFICA** | NVIDIA GTX 970 / AMD R9 290 o posterior |
| **PROCESADOR** | Intel i5-4590 o posterior |
| **MEMORIA RAM** | 8GB o superior |
| **PUERTO DE SALIDA** | Salida de vídeo compatible con HDMI 1.3 |
| **PUERTOS DE ENTRADA** | 3 puertos USB 3.0 más un puerto USB 2.0 |
| **SISTEMA OPERATIVO** | Windows 7 SPI de 64 bit o superior |

*Tabla 1. Requisitos mínimos para soporte en VR [22]*

### 3.2 Elementos de Software

Según explica Boo Gustem [3], dentro de cualquier sistema de realidad virtual aparecerán los siguientes elementos de software:

- **Modelo geométrico 3D**

Para que el usuario pueda viajar a través de su mundo virtual e interactuar con él, necesita de una representación en 3D de este mundo, que haga los cálculos de renderizado, generación de audio, que permita calcular el comportamiento de las colisiones, etc.

- **Software de tratamientos de datos de entrada**

Este se encargará de recoger la información capturada por los periféricos de entrada a nuestro ordenador. A base de ejemplo, la posición inicial y la orientación de la cabeza del usuario deben transformarse para poder expresarlas en un sistema de coordenadas que la aplicación pueda leer y posteriormente filtrar para evitar fallos.

- **Software de simulación física**

Los módulos que forman parte de la simulación física hacen referencia a aquellas alteraciones que tienen lugar en la representación del mundo virtual y que vienen producidas a través de acciones que provoca el usuario o el propio sistema. Así pues, si el usuario inicia una acción en la que se procede a empujar un objeto, el software de recogida de datos indicaría la voluntad del usuario de realizar dicha acción y el software de simulación física se encargaría de aplicar la transformación geométrica necesaria al objeto del modelo 3D que lo representa.

- **Software de simulación sensorial**

Este software será el responsable de transmitir a nuestros sentidos los estímulos que permitan al usuario percibir el espacio virtual de la forma más realista posible, a través de los periféricos de salida. El elemento más básico se llevará a cabo a través de la simulación visual, a través de algoritmos de visualización en tiempo real del modelo. Los

algoritmos auditivos son igual de complejos puesto que se deben tener en cuenta las propiedades acústicas de cada uno de los elementos.

## 3.3 Soporte Físico del proyecto

### Oculus Rift

El equipo de realidad virtual que se ha utilizado para poder llevar a cabo el proyecto es el modelo Oculus Rift. Dentro de los tipos de dispositivos este es un visor inmersivo de escritorio, de manera que necesita de un ordenador para poder funcionar. Cuenta a su vez con dos sensores y dos controladores.

- **Visor**

El visor que nos presenta Oculus Rift es liviano y cómodo, con bandas ajustable en los laterales y en la parte superior de la cabeza. Las lentes también son ajustables para un mejor enfoque con un ángulo de visión de 110º.

*Figura 19 Visor Oculus Rift [21]*

Las lentes equipan tecnología OLED con una resolución de 2160x1200 pixeles y una tasa de refresco de 90 Hz. Además, cuentan con su propio sistema de audio con auriculares con sonido envolvente 3D.

Para su conexión y alimentación cuenta con un cable de 4 metros de longitud que sale de la parte posterior del casco y se conecta con el ordenador por el puerto USB y HDMI.

- **Touch Controllers**

En cuanto a los mandos, estos presentan un diseño muy ergonómico, adaptándose a la perfección a nuestra mano. Cuenta con dos botones y un joystick analógico que se accionan desde el pulgar, y dos gatillos sobre el mango para simular el movimiento de agarre y puntero. Funcionan con pilas AA.

*Figura 20 Oculus TouchController[21]*

Estos controladores presentan sensores que permiten situar la posición de los dedos además del gastillo que ya hemos comentado, que simula la acción de agarre.

- **Sensores**

Los sensores de Oculus Rift, también llamados Oculus Constellation, detectan el movimiento de nuestro cuerpo mediante el rastreo por infrarrojos como se ha mencionado anteriormente. Se usan mínimo dos sensores, que se disponen en los laterales de nuestro ordenador para tener un ángulo de visión de 180º del espacio de rastreo. Se podría obtener un rastreo de 360º colocando un sensor en la parte superior de la zona de rastreo. Se conectan al ordenador mediante un cable USB.

*Figura 21. Sensor de posición de Oculus [21]*

## Ordenador de Sobremesa MSI GE73 Raider RGB 8RE

| ESPECIFICACIONES TECNICAS | |
|---|---|
| **Procesador** | |
| **Nombre** | Intel Core i-7 |
| **Arquitectura** | 64 bits |
| **Núcleos** | 6 núcleos |
| **Frecuencia base** | 2.2 GHz |
| **Frecuencia Turbo** | 4.1 GHz |
| **Chipset** | Intel HM370 |
| **Tarjeta Gráfica** | |
| **Nombre** | NVIDIA GeForce GTX 1060 |
| **Memoria RAM** | 6GB GDDR5 |
| **Características generales** | |
| **Memoria** | 16GB DDR4 |
| **Almacenamiento** | 256GB SSD, 1TB HDD |
| **Resolución de pantalla** | 3420x2160 píxeles |

*Tabla 2. Especificaciones del Ordenador MSI*


*Figura 22. Ordenador MSI*
*Fuente:MSI.com*

# CAPÍTULO 4. MODELADO DEL TORNO CMZ T-360

## 4.1 Torno CMZ T-360

Nacida en el año 1945 en unos bajos en la calle Aldatza en Zaldibar, la empresa CMZ fue fundada desde un principio con carácter familiar de la mano de Florencio Zumárraga. En un principio se centrarían en la fabricación de máquinas regruesadoras utilizadas en carpintería, para luego comenzar con la fabricación de bordoneras accionadas a mano y seguidamente con las limadoras L-350 con mucho éxito durante los años 50.

Tras sobrevivir a la crisis de la estabilización (1958), CMZ se consolida como uno de los fabricantes más importantes en maquinaria de uso industrial y se abre al mercado extranjero en Alemania y Colombia (1961).

En 1970 la empresa CMZ, atendiendo a que los tornos ocupaban la mayor parte de la demanda del mercado (alrededor del 80% en aquella época), se lanza a la producción de esta nueva máquina, dejando de lado la fabricación de limadoras. Es aquí cuando surgen los tornos T-360, T-410 y T-500.

Este tipo de tornos pertenecen a la categoría de tornos paralelos o mecánicos. En la actualidad este tipo de tornos se han ido sustituyendo por tornos copiadores, revólver, automáticos y de CNC. Por lo que hoy en día estos tornos se utilizan para realizar tareas menos importantes, como operaciones puntuales o en talleres de aprendices y en escuelas de forma didáctica como es en nuestro caso.

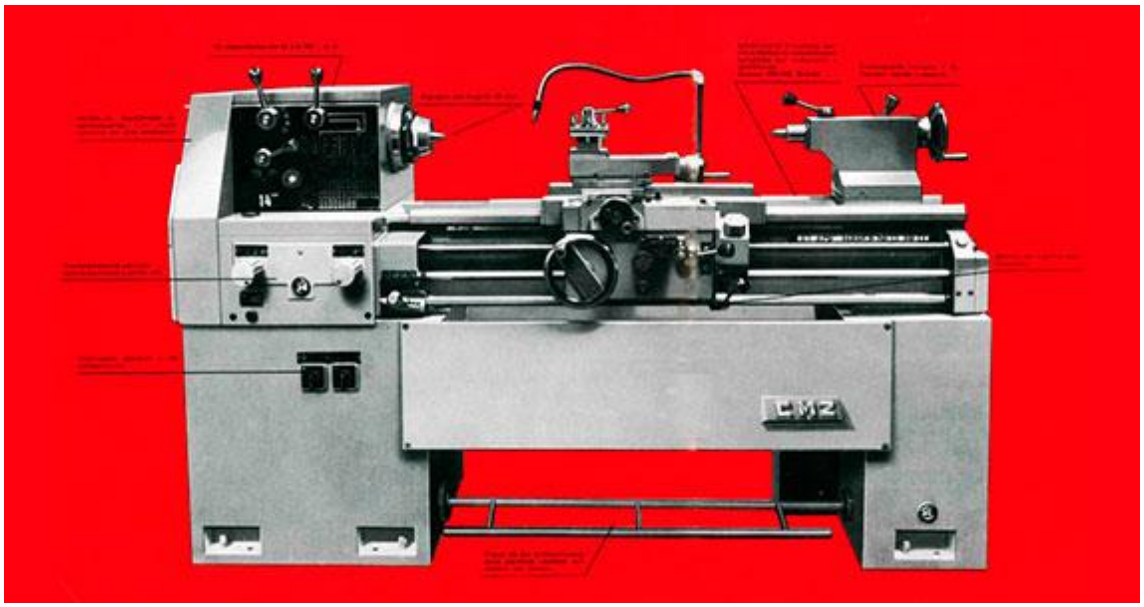



*Figura 23. Torno CMZ T-360 [23]*

### 4.1.1 Aplicaciones

El torno paralelo es una máquina herramienta de corte circular continuo que permite el mecanizado del material mediante el avance de una herramienta de corte monofilo. En este tipo de operaciones, el movimiento de corte lo lleva la pieza, la cual se encuentra anclada a la máquina y se hace girar a unas determinadas revoluciones. Por su parte, la herramienta avanza en sentido longitudinal a la pieza, mecanizando el material mediante el arranque de viruta.

Podemos producir diferentes tipos de superficie de revolución ya sean cilíndricas, cónicas, así como roscas, etc. Es por ello por lo que los tornos representan una de las máquinas de herramientas más básicas y esenciales en mecanizados por arranque de viruta.

Entre las operaciones que se pueden llevar a cabo, se explicarán brevemente las dos que se llevan a cabo dentro del programa de prácticas del departamento de fabricación.

- **Cilindrado**

Mediante esta operación se consiguen superficies cilíndricas mediante dos tipos de cilindrado: cilindrado externo (figura 26) y cilindrado interno. Para ello la herramienta monofilo seguirá un movimiento de avance en dirección paralela al eje de la pieza.

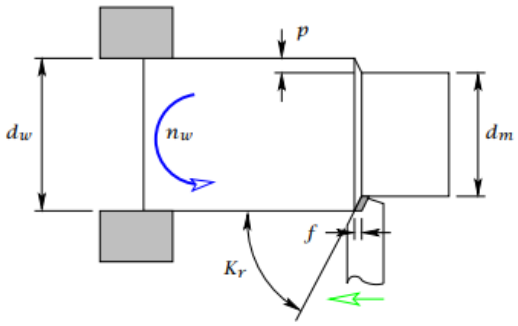



*Figura 24. Operación de cilindrado [24]*

***p***, profundidad de corte en el material.

***$d_m$***, diámetro mecanizado.

***$d_w$***, diámetro sin mecanizar.

*$K_r$*, ángulo de filo de corte.

***f***, longitud de avance de la herramienta por cada revolución de la pieza.

***$n_w$***, velocidad de giro de la pieza

- **Refrentado**

Esta operación se emplea para producir superficies planas en la pieza. En este caso, la herramienta sigue un avance en dirección perpendicular al eje de giro de la pieza. Suele realizarse en el extremo de la pieza, sin embargo cuando se produce en alguna sección interior, esta pasa a llamarse tronzado (figura 27).

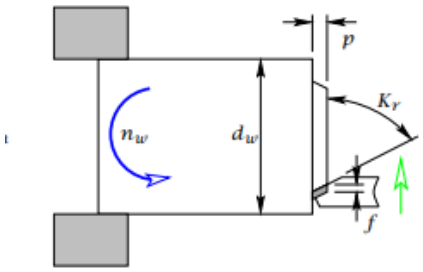



*Figura 25. Operación de refrentado [24]*

En este caso la velocidad de corte no será continua, puesto que nos estamos desplazando radialmente hacia el interior de la pieza, el diámetro disminuye y con este la velocidad lineal.

*Figura 26. Ejemplo real de cilindrado*

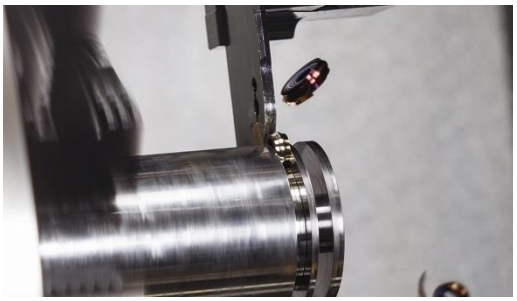


*Figura 27. Ejemplo real de tronzado*

*Figura 28. Ejemplo de refrentado*

### 4.1.2 Componentes principales del Torno CMZ T-360

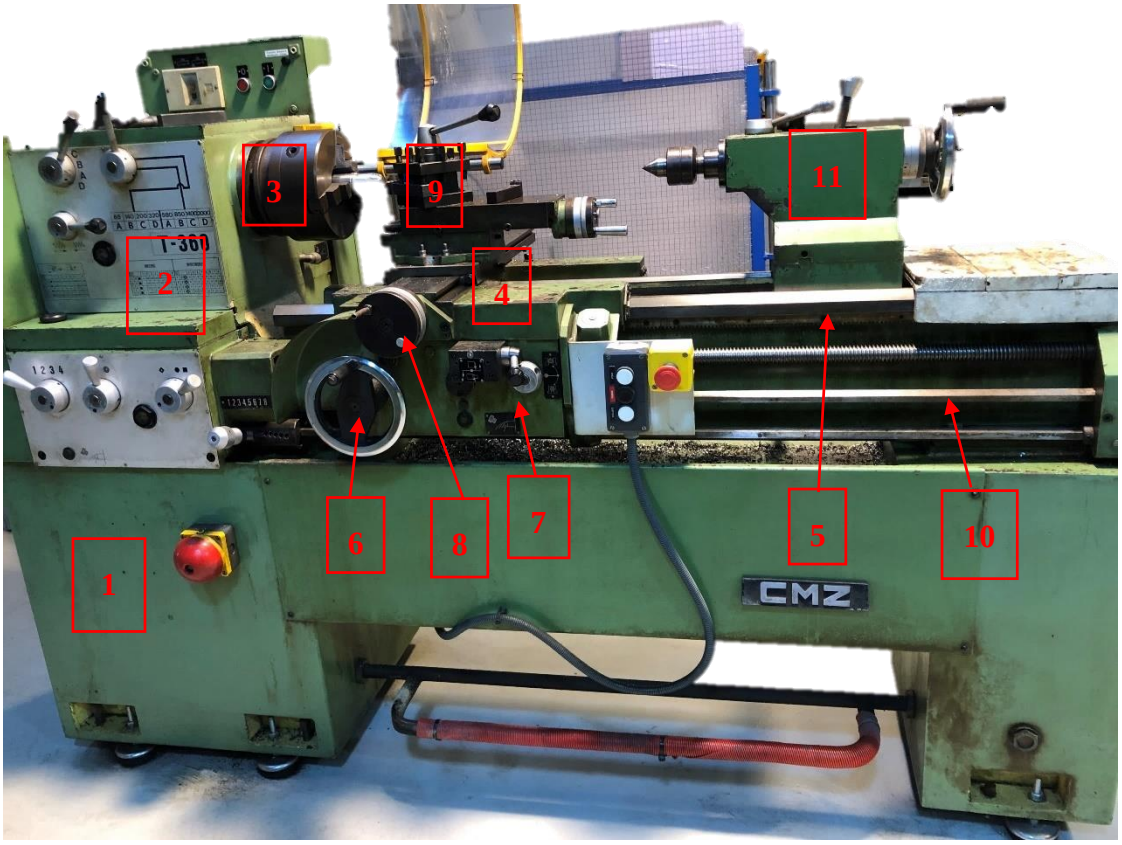



*Figura 29. Torno T-360 ELDI*

La **Bancada [1]** es el elemento en el que se asientan los diferentes componentes y carros del torno. Es el cuerpo de mayor tamaño con diferencia y el de mayor peso (fabricado en hierro gris o hierro fundido). En la parte superior presenta dos guías sobre las que se asientan otros carros para poder realizar el mecanizado.

El **cabezal [2]** se fija a la bancada y suministra la energía al **husillo [3]**. El husillo, a través del plato de agarre, sujeta la pieza de manera que ésta pueda rotar solidaria a él. Existen diferentes palancas de selección de velocidades de giro manuales. El husillo presenta también mandriles o garras ajustables para poder adaptar diferentes piezas de trabajo.

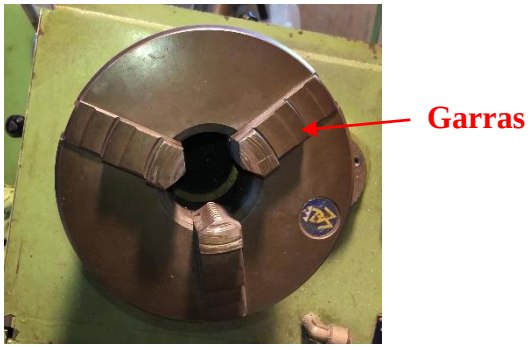



*Figura 30. Plato de garras*

El **carro longitudinal [4]** se desliza a lo largo de las **guías [5]** de la bancada en la dirección "x", desplazando a la herramienta. El movimiento del carro se acciona manualmente a través del **volante del carro longitudinal [6]**. El carro longitudinal también se desplaza de forma automática mediante el accionamiento de la **palanca de cilindrar y refrentar [7]** en función de los avances seleccionados a través de los mandos de la bancada.

El **carro transversal** (figura 31) está apoyado sobre el carro longitudinal. Este a su vez desplaza la herramienta en sentido transversal al husillo. Para accionar manualmente el desplazamiento del carro transversal se utiliza el **volante del carro transversal [8]**. Además, este carro se desplaza automáticamente en las operaciones de refrentado mediante la palanca de refrentar. El avance que sigue se fija mediante los mandos de avance siendo 0.5x el avance para cilindrado.

*Figura 31. Carro transversal*

El **carro portaherramientas [9]** se encuentra montado sobre el carro transversal, y es el carro que sirve de soporte a la herramienta. Este puede desplazarse en la misma dirección que el carro longitudinal a través del **volante del carro portaherramientas**.

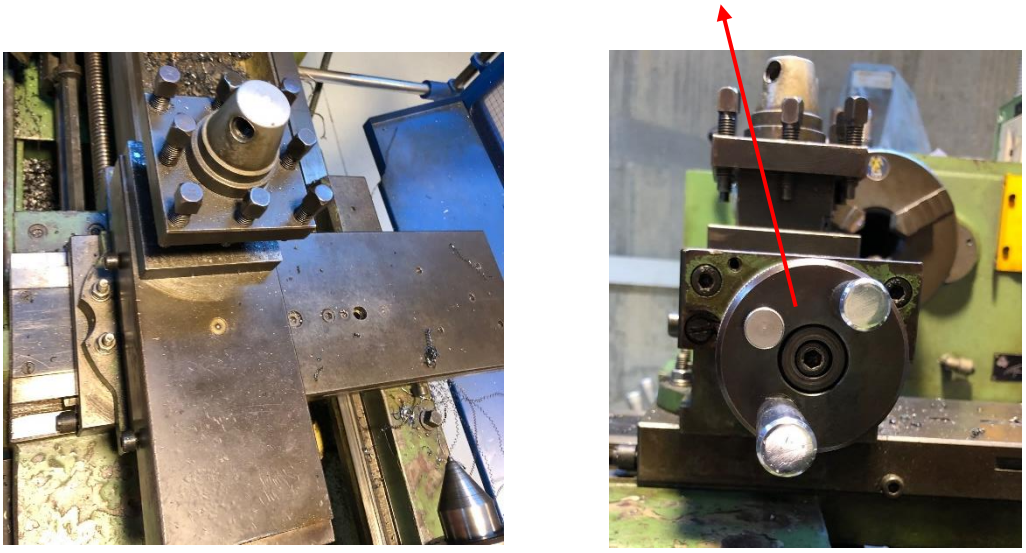


*Figura 32. Carro portaherramientas*

La **barra de avance [10],** accionada por el cabezal se mueve al accionar el husillo y permite el desplazamiento del carro longitudinal y el carro transversal por medio de engranajes.

El **carro del contrapunto [11]** o contra cabezal, también va montado sobre la bancada y se desliza por las guías para poder retener la pieza por el otro extremo, para evitar posibles desnivelaciones o reducir los esfuerzos en el husillo. Se puede fijar a la bancada mediante la **manivela de bloqueo del contrapunto**. La caña del contrapunto también se puede desmontar mediante la **manivela de la caña del contrapunto**, que nos permite sustituir la punta del contrapunto por la broca de centrar para después sujetar la pieza. En cuanto al eje del contrapunto, se desplaza mediante el **volante del contrapunto**. En su extremo se coloca la **broca de centrar** que permiten taladrar la pieza para posteriormente sujetar la pieza con el **contrapunto**.

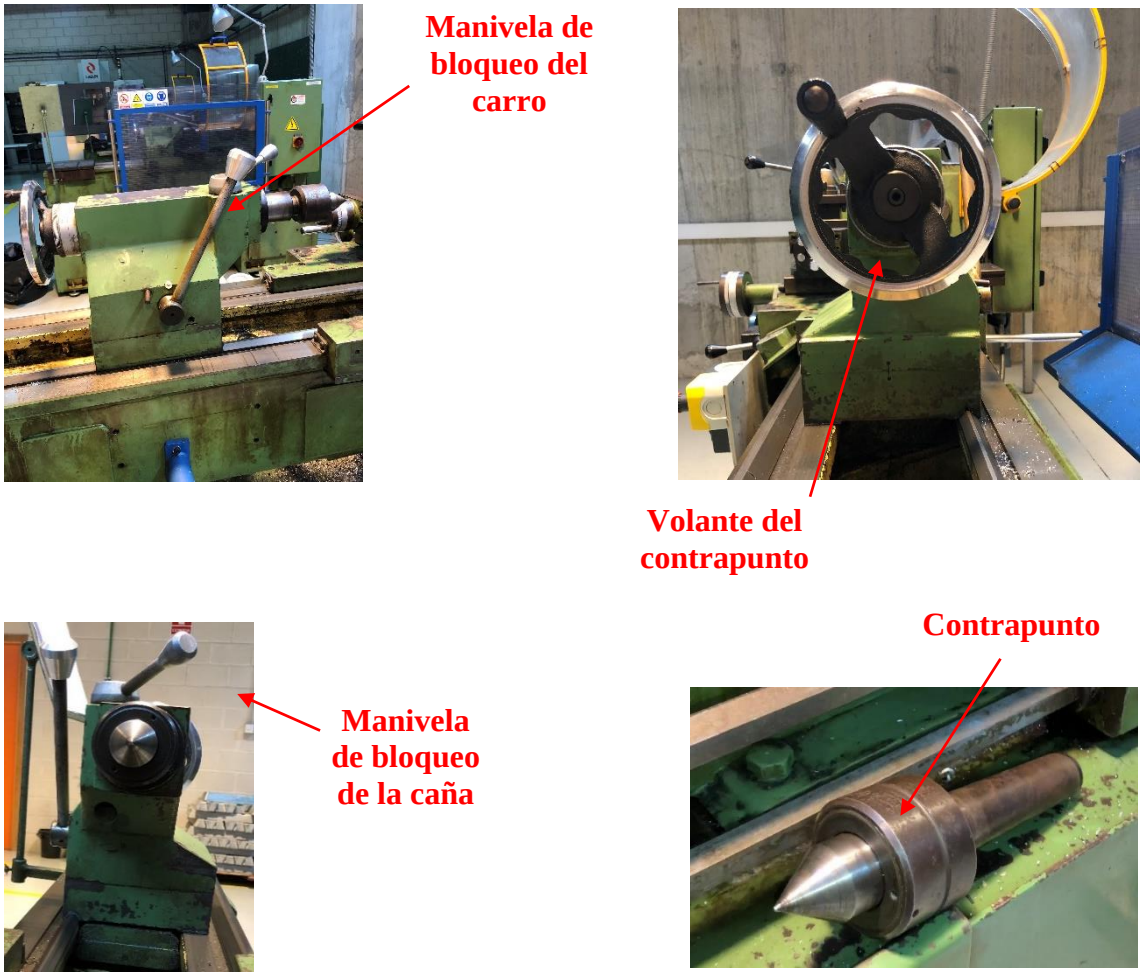



*Figura 33. Carro del contrapunto*

En cuanto a la **herramienta de corte**, se utilizan herramientas monofilo y según el sistema de sujeción de la plaquita y el mango, pueden ser herramientas de plaquita soldada o sujeta mecánicamente (figura 34). Están fabricadas a partir de materiales muy resistentes a la abrasión, con elevada dureza y resistencia mecánica a esfuerzos de flexión y compresión y a trabajar bajo elevadas temperaturas (Aceros al carbono, aceros rápidos, materiales cerámicos, etc.)

*Figura 34. Herramienta de corte con plaquita sujeta mecánicamente*
*Fuente: Sandvik Coromant - herramientas y soluciones para el mecanizado*

## Mandos de selección de avances y velocidades

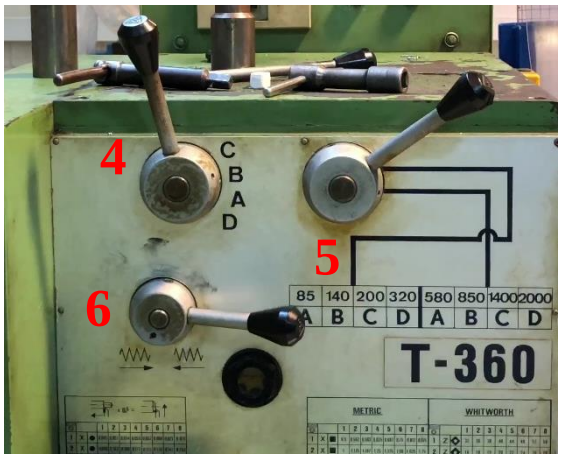





*Figura 35. Mandos de la bancada*

1. Mando de selección de avances.
2. Mando del selector de roscas.
3. Mando de la barra de cilindrar y de roscar.
4. Mando de velocidades.
5. Mando de velocidades lentas y rápidas.
6. Palanca de inversión de avance.
7. Mando de la caja Norton.

## Botones e interruptores

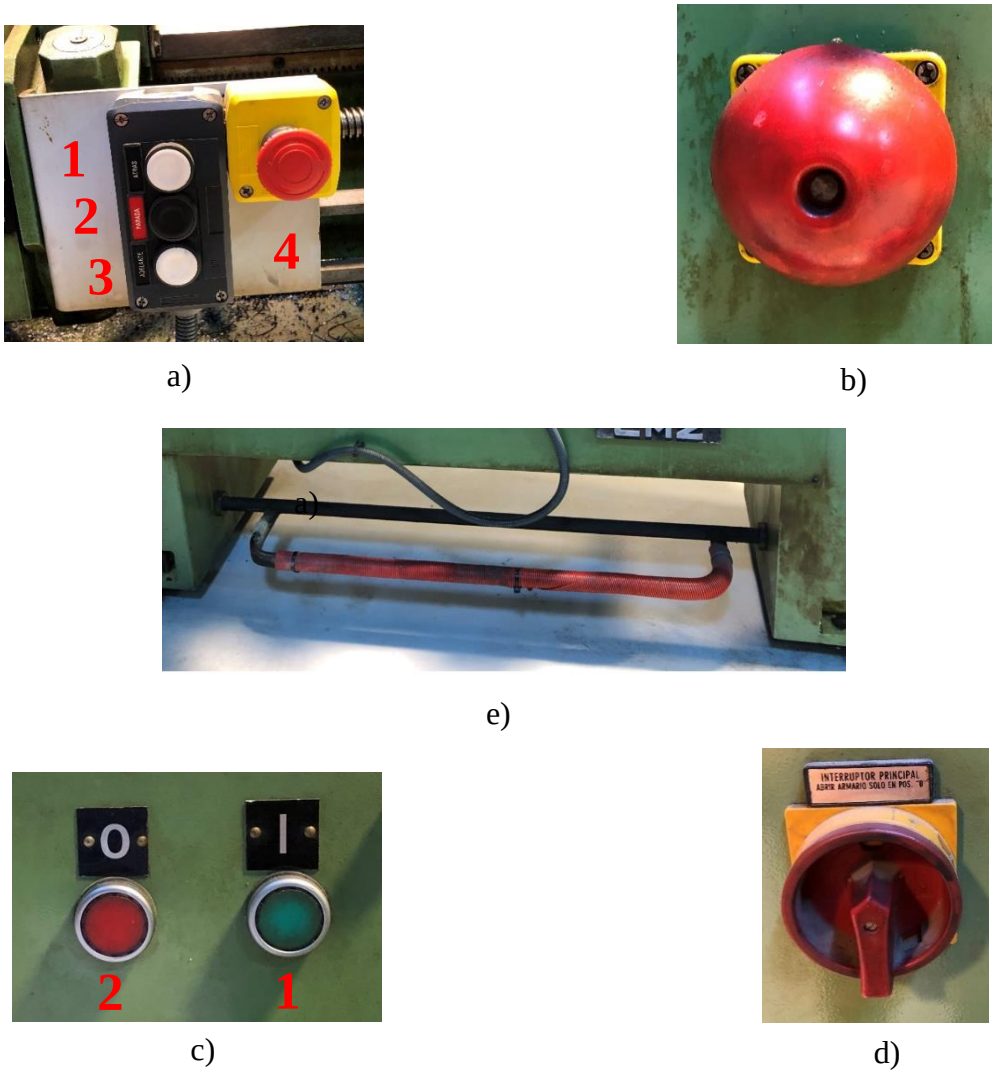



*Figura 36. Botones e Interruptores principales del torno*

a) Botonera del carro longitudinal.
    1. Atrás (invierte el sentido de giro del Husillo).
    2. Parada (paraliza la rotación del Husillo).
    3. Adelante (inicia el sentido de giro del Husillo).
    4. Freno de Emergencia (paraliza la rotación y desactiva la corriente).
b) Seta de Emergencia (misma función que el freno de emergencia).
c) Electricidad (permite el suministro de energía al cabezal).
    1. Activar corriente.
    2. Desactivar corriente.
d) Contacto general de la corriente (permite el suministro de energía de la toma general al torno).
e) Freno de pie (frena el Husillo).

## 4.2 SolidWorks

Para llevar a cabo el modelado de nuestro torno se utilizará SolidWorks. Este es un software de diseño tipo CAD (computer aided desing) que utiliza un entorno gráfico basado en Microsoft Windows, de manera que se pueden realizar de forma sencilla pero eficaz, modelos en 3D, planos en 2D y ensamblajes.

La primera versión de este software nace en el año 1995, convirtiéndose en la primera tecnología CAD que podía ser capaz de ejecutarse en Windows. Como es de esperar, se iniciaría con algunas limitaciones, especialmente en el aspecto del ensamblaje, sobre todo en computadores de gama media.

Estos problemas se irían solucionando y para el año 1996 se incorporará el módulo COSMOS/Works con el que se permite al propio usuario realizar un análisis de fatiga sin tener que recurrir a otros departamentos de análisis estructural que incorporaban profesionales que usaban software de análisis de elementos finitos (ANSYS).

En el año 1997 se incorporan nuevas funcionalidades como el diseño de chapa, extruir texto sobre la superficie plana de los diseños etc. También se anunciaría la compra por parte de Dassault Systemes S.A. que adquirió SOLIDWORKS (empresa fundada por Jon Hirschtick en 1993) por 310 millones de dólares.

Para 1998, este software está a la altura de ser el mejor de su campo, aumentando sus ventas enormemente e iniciando una “guerra” de marketing contra su rival en este campo: Autodeks.

*Figura 37. Logotipo SolidWorks*

### 4.2.1 Aplicaciones

En la actualidad existen alrededor de 210.000 empresas conocidas que utilicen este software a la hora de desarrollar su propio producto y que forman parte de industrias diferentes como: Aeroespacial y defensa, servicios a empresas, construcción, ciudades y territorios, productos envasados y venta al por menor, energía y materiales, alta tecnología, hogar y estilo de vida, maquinaria industrial, ciencias de la salud, naval y alta mar, transporte y Movilidad.

- **Diseño/Ingeniería**

Aquí se encontrarán intuitivas soluciones a la hora de diseñar proyectos en 3D y 2D mediante los productos CAD 3D y CAD 2D. También se tienen herramientas para el diseño eléctrico para crear sistemas eléctricos integradas a través de SOLIDWORKS Electrical. Otra novedosa herramienta que presenta es SOLIDWORKS MBD para realzar anotaciones en 3D incluyendo los datos de modelos 3D (tolerancias, cotas, lista de materiales...).

- **Control/Gestión**

Se encuentran herramientas que reutilizan los datos de CAD en 3D para mejorar la comunicación técnica dentro de la empresa.

- **Simulación**

Por otro lado, existen herramientas de simulación que permiten conocer la interacción el diseño que hemos creado para poder evaluarlo y tomar decisiones que vayan a favor de mejorar la calidad del producto. Esto ayudará a reducir costes de prototipos y pruebas físicas. SOLIDWORKS Flow Simulation, SOLIDWORKS Plastics y SIMULIAworks son algunos de los productos que presenta esta línea.

- **Fabricación/Producción**

Optimización de la comunicación entre los diferentes departamentos de producción para identificar errores de fabricación y poder mitigarlos. SOLIDWORKS CAM, SOLIDWORKS Inspection y 3D EXPERIENCE Marketplace son algunas de las soluciones.

- **Marketing/Ventas**

Con el fin de dirigirse a los clientes objetivos, Solidworks ofrece una solución para cumplis con los planes de comercialización desde su concepto hasta la etapa final de venta mediante SOLIDWORKS Composer o Visualización.

### 4.2.2 Interfaz de usuario

Para empezar a desarrollar nuestro modelo, se planeará el método de creación de este. Una vez realizado, se comenzará con su desarrollo. Al iniciar el Software, la primera ventana (figura 38) que aparece permitirá elegir entre el tipo de proyecto que queremos llevar a cabo.


*Figura 38. Ventana nuevo proyecto de SolidWorks*

Las piezas son los componentes más básicos de un proyecto, estos darán lugar a la formación de ensamblajes que a su vez podrán estar formados de otros ensamblajes

denominados subensamblajes. Por otro lado, los dibujos permiten representar de forma bidimensional las representaciones en 3D de estos ensamblajes.

#### *4.2.2.1 Piezas*

Para comenzar el desarrollo del modelado, se creará una pieza. En la ventana emergente encontramos diferentes paneles (figura 39).

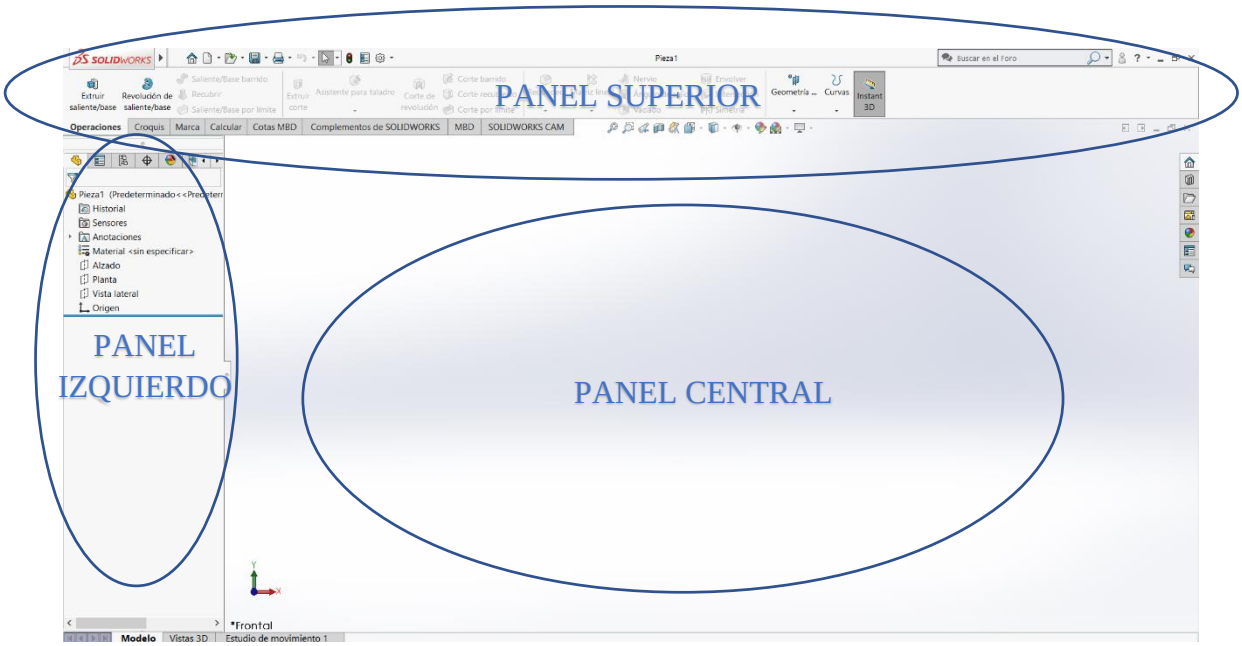



*Figura 39. Ventana principal para proyecto tipo Pieza*

En el panel central se encuentra la zona de gráficos. En esta sección se puede crear y manipular una pieza, del mismo modo que se hará en el caso de trabajar con ensamblajes o dibujos.

En la parte del panel izquierdo se destacan los siguientes aspectos:

- La ventana del **Gestor de diseño del Feature Manager**. Se muestra una lista de la estructura que tiene nuestra pieza, del mismo modo para nuestro ensamblaje o dibujo. Seleccionando uno de los elementos se puede editar el croquis subyacente, así como cualquier operación realizada dentro de la pieza.

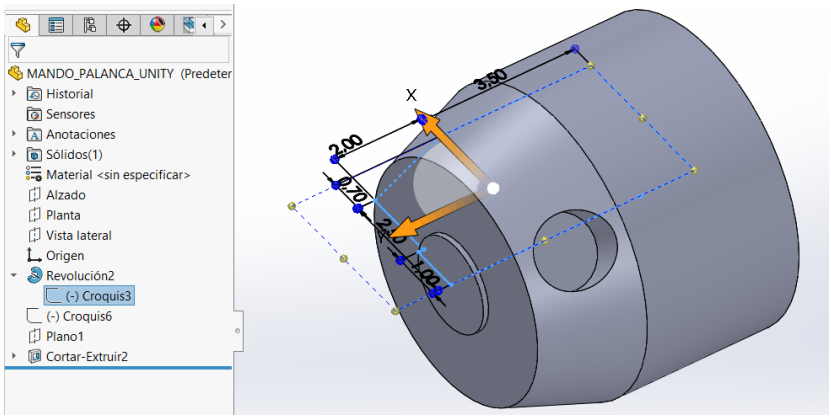



*Figura 40. Ventana Feature Manager*

Al seleccionar el croquis subyacente a la operación de revolución (mediante la cual se ha generado dicha pieza) se puede acceder a él para poder editarlo. También permite cambiar el estado de supresión de la operación o del componente.

- La ventana **Property Manager**. Permite configurar las propiedades y otras opciones de los diferentes comandos que se pueden seleccionar.

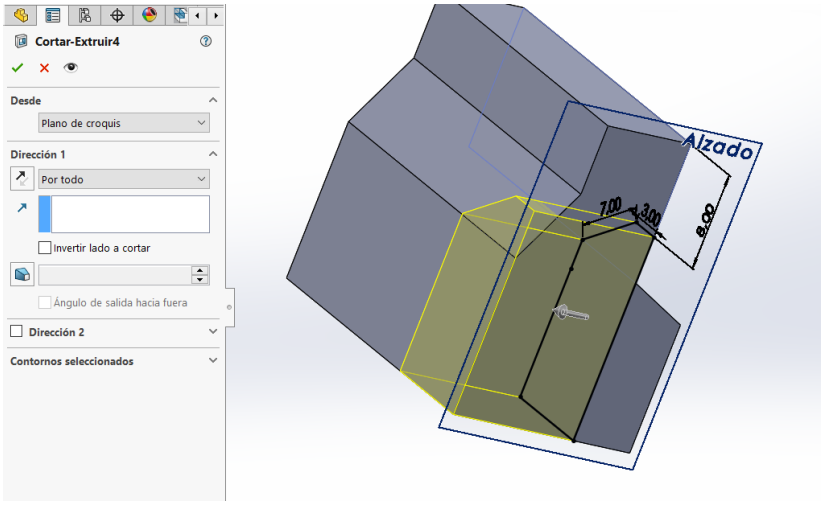



*Figura 41. Ventana Property Manager*

Desde la ventana del Gestor de diseño del FeatureManager se accede a editar la operación de **Extruir corte** que se ve en la ilustración. De esta manera se abrirá la ventana Property Manager que nos permitirá editar dicha operación sobre la pieza. Al tratarse de un corte, se puede seleccionar la distancia de extrusión y la dirección de esta.

Si nos fijamos ahora en el panel superior, se encuentra lo que se conoce como administrador de comandos. El administrador de comandos es una barra de herramientas sensible al contexto que actualiza dinámicamente en función de si estamos trabajando sobre una pieza, ensamblaje o dibujo.

En el caso de estar trabajando sobre una pieza, las herramientas que se pueden destacar dentro de esta sección son: la pestaña de operaciones y croquis.

- En la pestaña de **Croquis**, se introducen todas las herramientas necesarias para iniciar el diseño de la pieza en el plano central de la aplicación.


*Figura 42. Pestaña de croquis*

Existen diversas formas de crear un croquis, pero todos constan de los siguientes elementos:

Un **Origen**. La mayoría de los croquis se empiezan por el origen de coordenadas para tener una referencia

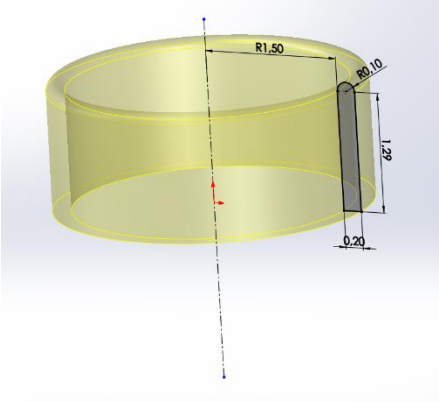



*Figura 43. Ejemplo de croquis para operación de revolución*

En la figura 43 se observa una pieza generada mediante la operación **Superficie de revolución** cuyo eje pasa por el origen. En los croquis también se utilizan líneas constructivas que se suelen introducir para generar este tipo de geometrías.

**Plano**. Para crear un croquis se debe seleccionar un plano, ya sea un plano estándar o agregado por el propio usuario. En la ilustración se ve como, a partir de un croquis realizado en un plano auxiliar, este se proyecta sobre el sólido para generar dicha superficie a partir de la operación **Envolver** (figura 44).

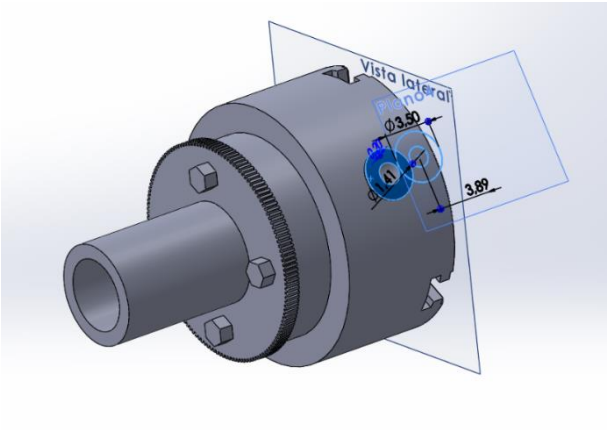



*Figura 44. Ejemplo de croquis sobre plano agregado*

**Cotas.** Estas definen el tamaño y la forma de la pieza. Pueden existir cotas entre entidades como longitudes o radios. SolidWorks diferenciará entre cotas conductoras y cotas conducidas.

— Las **cotas conductoras** (figura 45) permiten modificar el tamaño del modelo directamente. Por ejemplo, al modificar la altura como se muestra en la ilustración, se observa como la pieza se estiraría de manera desproporcionada puesto que la recta vertical que pasa por el origen no está acotada.

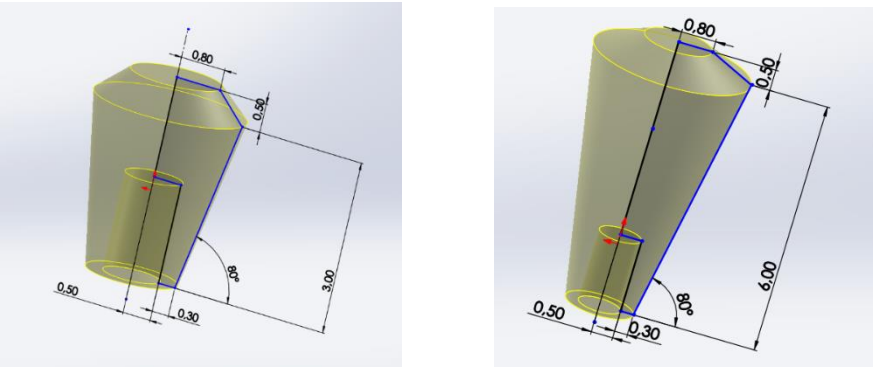



*Figura 45. Cotas conductoras*

— En cuanto a las **cotas conducidas** (figura 46). Se pueden usar como referencia con fines informativos. Estas cotas cambian si se modifica el valor de las cotas conductoras o las relaciones del modelo.

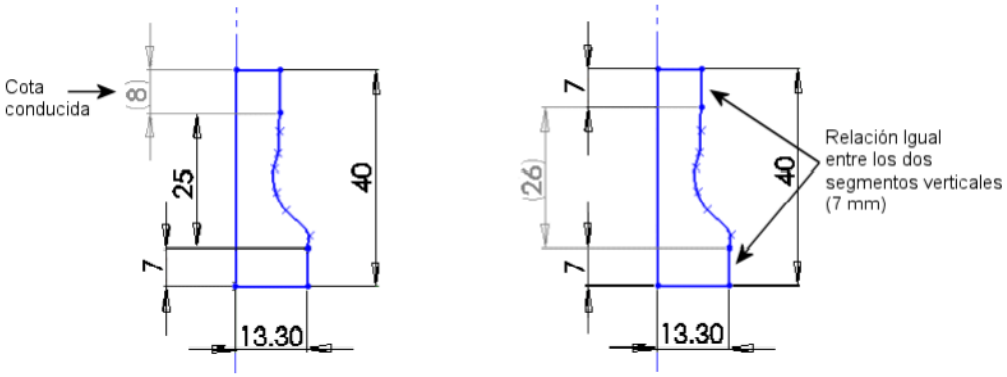



*Figura 46. Cotas conducidas*

**Relaciones**. Las relaciones (Tabla 3) establecen las relaciones geométricas entre entidades presentes en el croquis. Estas pueden ser relaciones de tangencia, perpendicularidad, coincidencia, igualdad, concentricidad, etc.

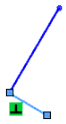
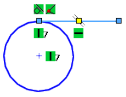
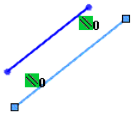
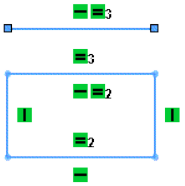
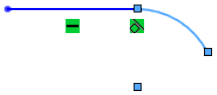
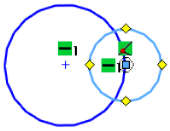
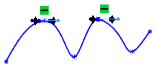
| Relaciones | Iconos | Relaciones | Iconos |
|---|---|---|---|
| Perpendicular |  | Verticales, horizontales, de intersección y tangentes |  |
| Paralelo |  | Horizontales, verticales e iguales |  |
| Horizontales y tangentes |  | Concéntrica |  |
| Horizontales y coincidentes |  | Horizontal |  |

*Tabla 3. Relaciones de croquis*

- Por otro lado se encuentra la pestaña de **Operaciones** (figura 47).

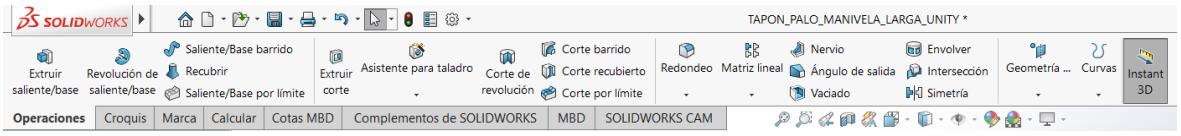



*Figura 47. Pestaña de operaciones*

Dentro de esta pestaña se introducen las posibles operaciones que podemos llevar a cabo para realizar el modelado de la pieza pieza. Estas se utilizan una vez que el croquis queda definido completamente. De esta manera se covierte el croquis en 2D en un sólido de 3D.

En cuanto a las operaciones que se pueden realizazar, se han visto ya algunas como **Extruir corte**, **Superficie de revolución** o **Envolver.** Estas forman parte de las denominadas **Operaciones basadas en croquis**.

**Extruir saliente/base**. Se agrega altura al croquis (figura 48) en la dirección perpedincular a este, aunque pueden especificarse otros parámetros como: ángulo de salida, dirección, plano desde donde se quiere realizar extrusión, etc

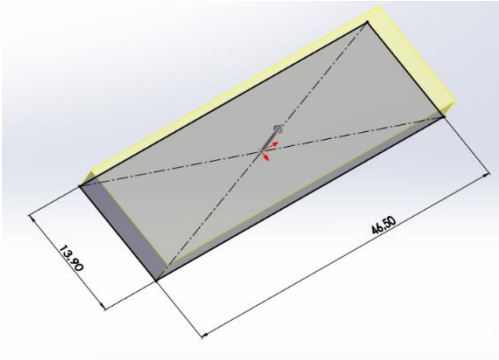



*Figura 48. Operación de extrusión*

**Barrido**. A partir de una sección se puede barrer dicha superficia a través de una trayectoria definida mediante una linea recta o una curva. En la figura 49 se muestra una pieza que forma parte del freno del husillo que se ha generado a partir de una sección circular (croquizada sobre el alzado) barrida a traves de un eje (croquizado anteriormente sobre la planta).

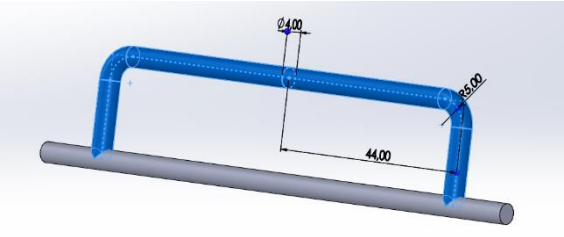



*Figura 49. Operación de barrido*

**Recubrir**. Mediante esta operación se consigue generar un sólido a traves de dos croquis situados en planos paralelos.

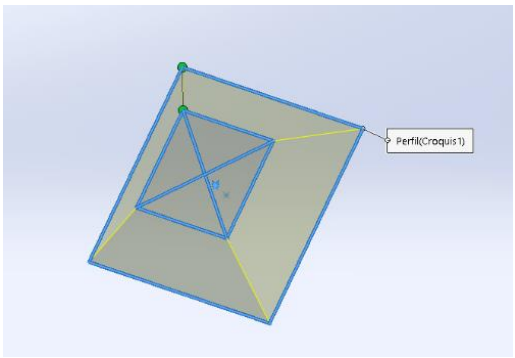



*Figura 50. Operación Recubrir*

**Asistente para taladro**. Útil a la hora de extruir o cortar roscas especificando tipo y tamaño.

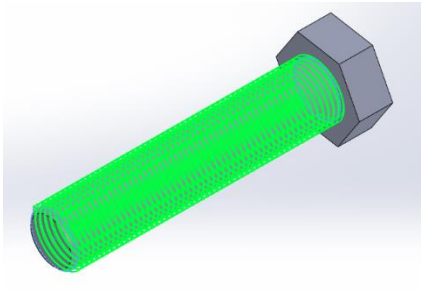


*Figura 51. Asistente para taladro*

Por otro lado se encuentran las **Operaciones aplicadas**. Estas operaciones se realizan sin necesidad de partir de un croquis, pero deben agregarse una vez se hayan realizado las operaciones basadas en croquis. Incluyen: chaflanes, redondeos o vaciados.

**Chaflan**. En la figura 52 se ve como el plato de garras presenta, en la parte frontal, dos chaflanes. Uno alrededor de la circuferencia exterior, y otro presente en una extrusion realizada en la cara frontal.

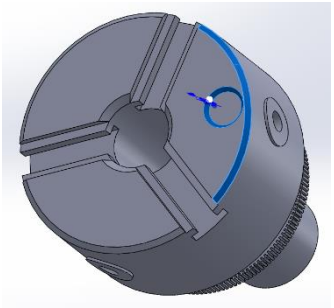


*Figura 52. Chaflán*

**Redondeo**. Mediante la operación de redondeo se crea una cara externa o interna redonda en la pieza. Se pueden redondear todas las aristas de una cara, conjuntos de caras, etc.

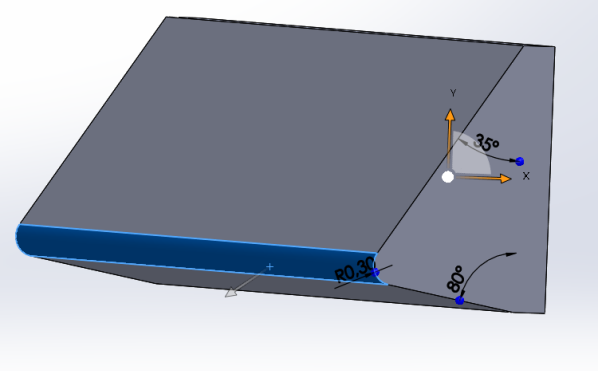



*Figura 53. Redondeo*

*4.2.2.2 Ensamblaje*

Como se ha dicho dicho antes, las piezas serán los bloques de construcción dentro de un ensamblaje. Para iniciar un ensamblaje, se debe inicair un nuevo proyecto y especificar que queremos realizar un ensamblaje.

Una vez que haya cargado la nueva ventana, en la sección del panel izquierdo, en la pestaña de PropertyManager, el software nos pedirá que seleccionemos los modelos para crear un nuevo ensamblaje.

A la hora de seleccionar el primer componente se debe seleccionar aquel que no se mueve respecto al resto. Este irá fijado en el origen del ensamblaje. En nuestro caso, la bancada (que a su vez será un subensamblaje) será el componente que deberá ir fijo dentro del ensamblaje, ya que es el elemento que sujeta al resto de componentes.

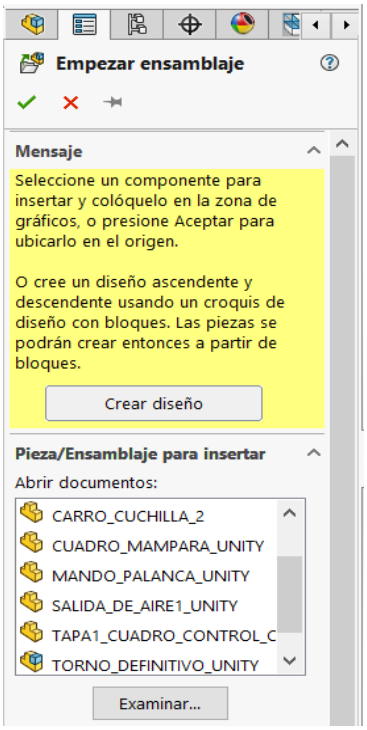



*Figura 54. Pestaña "Empezar ensamblaje"*

Una vez fijado el primer componente se puede ir introduciendo el resto de piezas que conformen el ensamblaje. Estos componentes se definen en relación al resto mediante relaciones de posición dentro del ensamblaje. Estas relaciones pueden ser de diferentes tipos: coincidentes, concéntricas o de distancia. De esta manera se definen los modos en que estos se desplazan o rotan respecto al resto de elementos.

A modo de ejemplo vamos a presentar la forma en la que introducimos el elemento carro longitudinal dentro del ensamblaje final de la bancada.

- **Paso 1. Cargar primer componente**. En este caso, el primer componente será la bancada junto algunos subensamblajes que irán fijos a esta. En caso de haber diseñado la bancada como una única pieza, a la hora de exportar el modelo a Unity, esto supondría un problema a la hora de aplicar texturas, materiales y otros componentes como se verá más adelante.

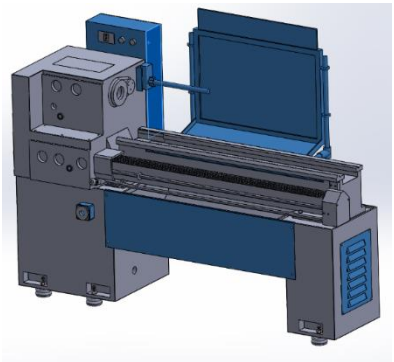
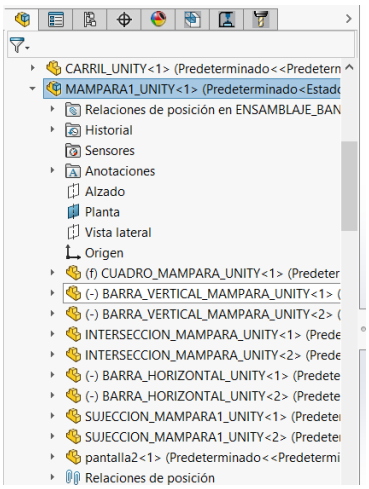


*Figura 55. Bancada del modelo*




Los subensamblajes que se introduzcan aparecerán dentro del Gestor de diseño del Feature Manager. Si se selecciona la mampara protectora presente en la parte posterior de la bancada y se selecciona dentro de esta pestaña, aparecerán las diferentes piezas que la componen.

Con el símbolo  se especifica que se trata de un ensamblaje y con  el simbolo que es una pieza.

Además aparecerán las relaciones de posición dentro del propio subensamblaje al final de dicho elemento.

*Figura 56. Despliegue Subensamblaje*

- **Paso 2. Cargar elementos adicionales**. En este caso se va a introducir uno de los elementos móviles que se asientan sobre la bancada: el carro longitudinal. Para ello habrá que ir al panel superior, al administrador de comandos y seleccionando la pestaña de Ensamblaje se encuentra la opción de *Insertar componentes*, donde se examinará en nuestro ordenador el componente que queremos insertar, ya sea una pieza única o un ensamblaje. Buscamos el carro longitudinal y lo insertamos.

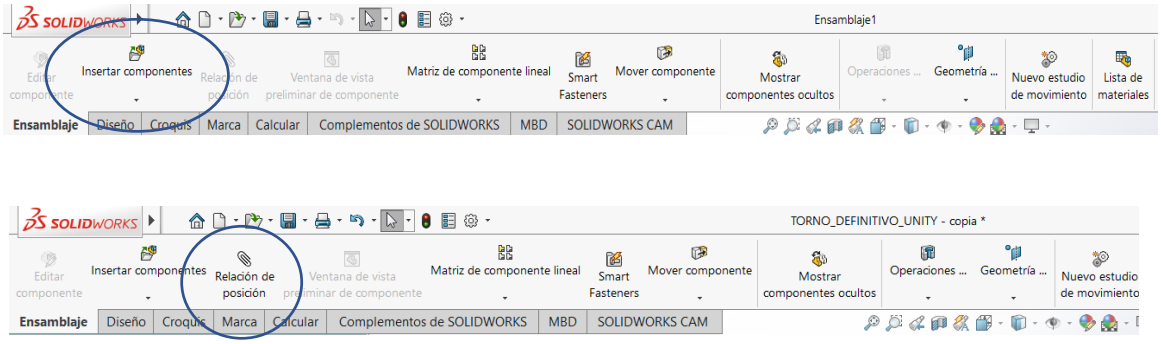



*Figura 57. Ensamblaje-administrador de comandos*

Ahora se establecen las relaciones de posición necesarias que definan su relación con la bancada. Para ello nos vamos otra vez al administrador de comandos y buscamos la opción de *Relación de posición*.

Pinchando se abrirá el PropertyManager para estableces las diferentes relaciones de posición. En nuestro caso queremos que el carro longitudinal vaya asentado sobre las guías de la bancada, de manera que este deslice sobre él. En este caso la primera relación que se introducirá será la de paralelismo entre los planos por los que desliza el carro.

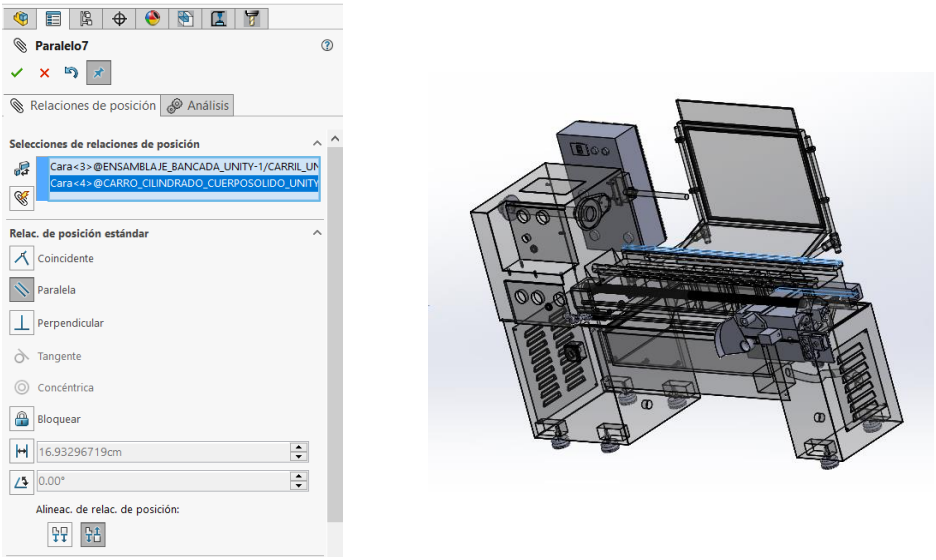



*Figura 58. Relaciones de posición SolidWorks*

Esta relación no es suficiente para definir la posición del carro, que como se ve en la figura 58 aparece dentro de la bancada, así que debemos seguir definiendo relaciones hasta que quede en la posición deseada. Al final se han necesitado tres relaciones de posición para definir correctamente la posición del carro. Concretamente dos relaciones de coincidencia. De esta manera el carro podrá deslizarse por la bancada con un grado de libertad.

Posteriormente a la creación de un ensamblaje vendría su representación en un dibujo, para lo que tendríamos que volver a abrir un nuevo proyecto y seleccionar la opción de *Dibujo*. En nuestro caso vamos a obviar la explicación de este procedimiento puesto que no se han realizado dibujos dentro del caso de estudio.

### 4.2.3 Modelo a escala real del Torno CMZ T-360

Una vez terminados todos los modelos que componen el Torno, se cargarían sobre la bancada procediendo de la misma manera que se ha hecho con el carro longitudinal en el apartado anterior. Estableciendo las relaciones de posición convenientemente para cada uno de ellos, el modelo final del Torno tendría el siguiente aspecto.

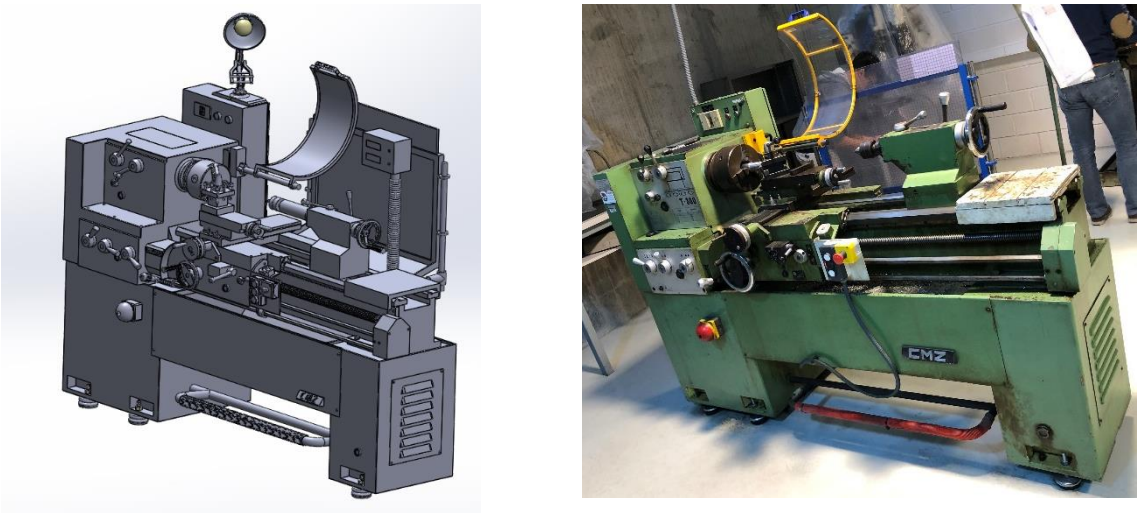



*Figura 59. Vista en perspectiva del modelo*

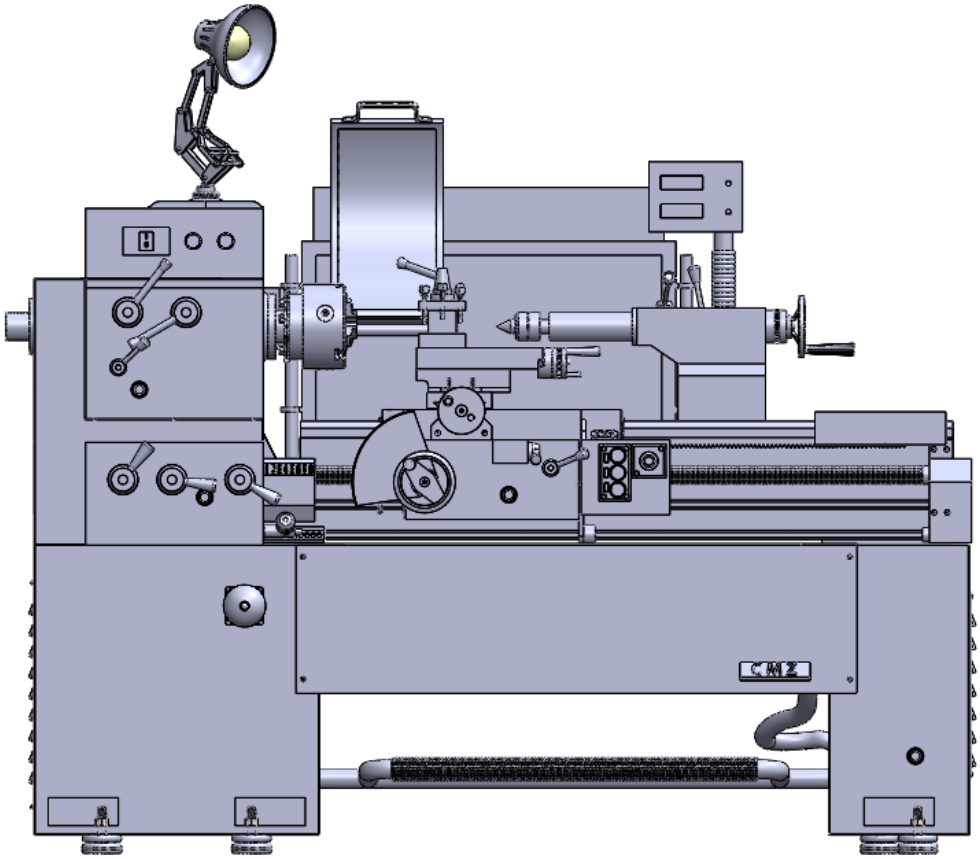


*Figura 61. Modelo 3D-Vista frontal*

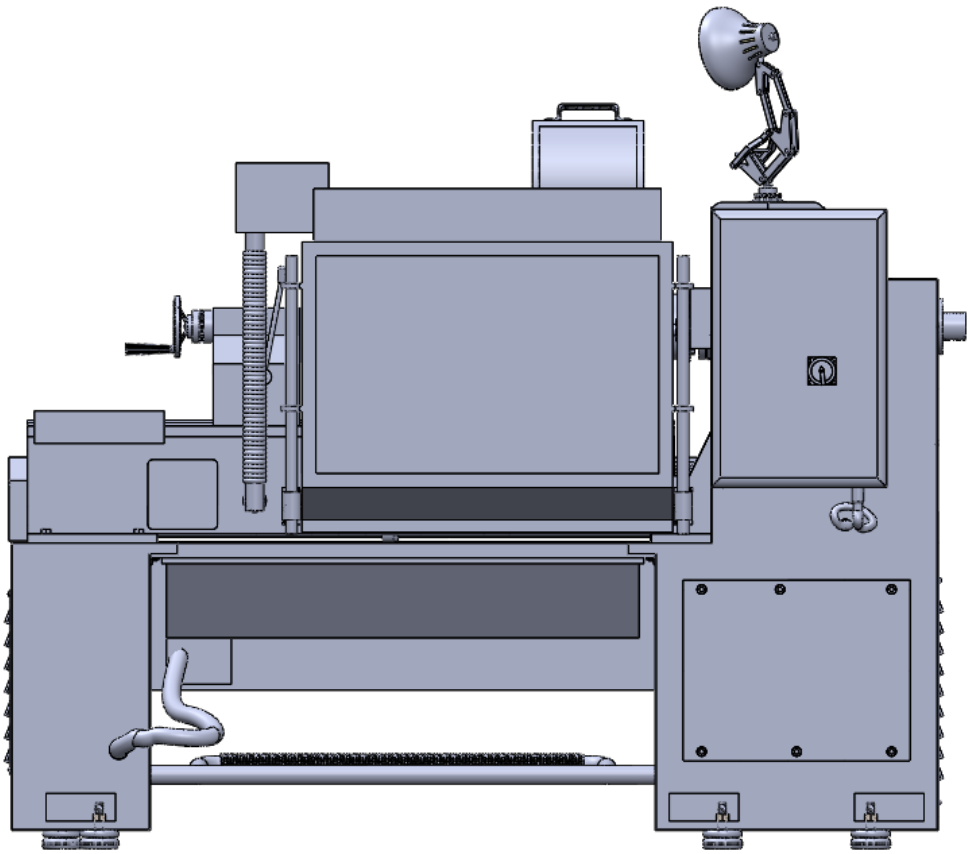


*Figura 60. Modelo 3D- Vista posterior*

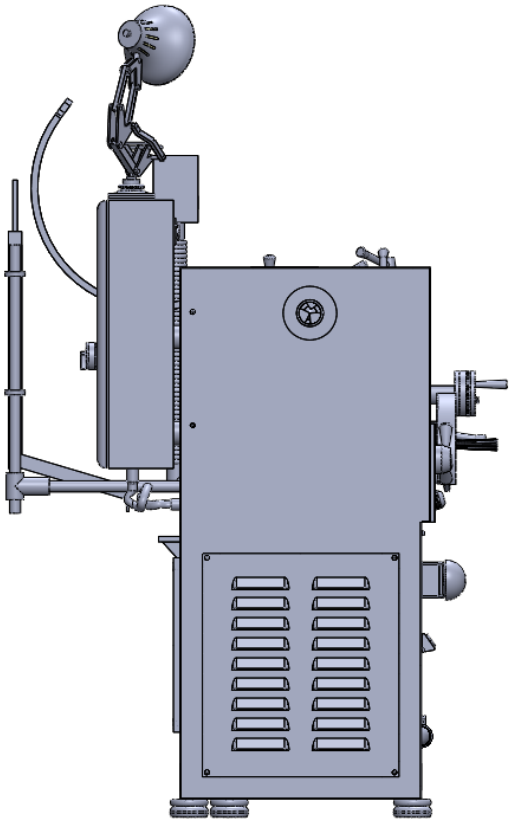


*Figura 62. Modelo 3D-Vista Perfil izquierdo*

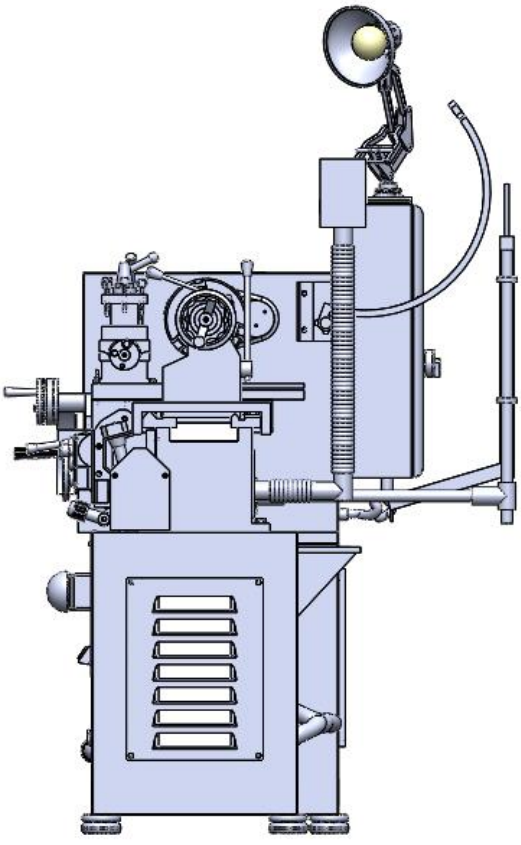


*Figura 63. Modelo 3D-Vista Perfil derecho*

Para el modelado del Torno no se tenían planos del mismo de manera que este estudiante realizó la toma de mediciones directamente utilizando un metro como herramienta de medición. Solo se tomaron nota de las cotas externas de los elementos de la máquina herramienta, de manera que estas tuvieran el aspecto y las dimensiones más ajustadas a la realidad para que el usuario mantuviera la concepción de estar utilizando el mismo Torno.

# CAPÍTULO 5. VIRTUALIZACIÓN DEL MODELO

## 5.1 Unity

El software que se empleará en el desarrollo del proyecto para poder interactuar con el modelo en realidad virtual será Unity.

Este software fue lanzado por primera vez en el año 2005, en la Conferencia Mundial de Desarrolladores de Apple en 2005, puesto que en un principio este software solo funcionaría para equipos de esta marca. Sin embargo, el éxito fue tal que se empezó a desarrollar para otras plataformas captando la atención de la mayoría de los desarrolladores que veían en este software una opción más accesible y con mayores prestaciones que otros softwares de gama alta que había para entonces.

La primera versión que se lanza es Unity 3 en 2010, seguida de Unity 3.5 que supondrá un gran éxito por la incorporación de nuevas mejoras como el renderizado HDR, sistema de partículas Shuriken, iluminación del espacio lineal, etc. Luego para el año 2012 sale otra nueva versión, Unity 4, donde se destaca su compatibilidad con Microsoft DirectX 11, la publicación en Linux y otras mejoras adicionales como el soporte de texturas 3D y la optimización del rendimiento y uso de memoria de UnityGUI. Posteriormente encontramos más actualizaciones con Unity 4.3 y la salida de Unity 5 en 2015.Unity 5.6 sería la última iteración dentro de Unity 5. En esta versión se produce una mejoría notable en las herramientas dedicadas a realidad virtual y realidad aumentada.

En 2017 la nomenclatura cambia y Unity crea la primera versión pública y sigue ofreciendo nuevas actualizaciones y mejoras hasta día de hoy con Unity 2020.

“Si buscas en Google "create a video game" recibirás 5.000 millones de resultados. De todos, Unity estará en 70 millones de ellos. Unreal Engine en 36 millones. Game Maker Studio en 2 millones. Va a ser difícil -no imposible- que a un principiante alguien le muestre otro camino. Mucho menos que le recomiende crear su propio motor” (Márquez, Raúl. 2020)

*Figura 64. Logotipo-Unity*

### 5.1.1 Interfaz de usuario

A la hora de crear un Proyecto aparecerá una primera ventana (figura 65) para seleccionar el tipo de proyecto que se quiere llevar a cabo y donde se quiere guardar dentro de nuestro ordenador.

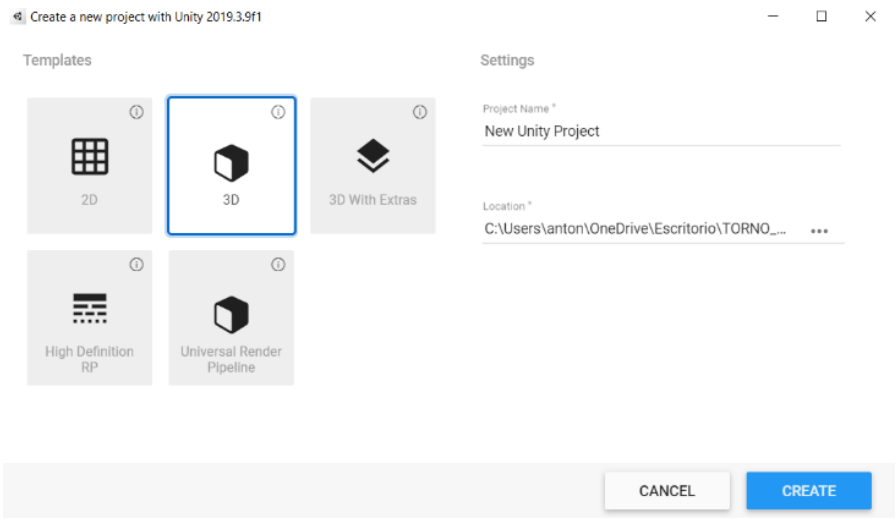



*Figura 65. Ventana de inicio de un proyecto-Unity*

En este caso elegiremos la opción 3D, por lo que seguidamente se abrirá la pantalla de la aplicación que tendrá una vista como la que se puede visualizar en la figura 66.

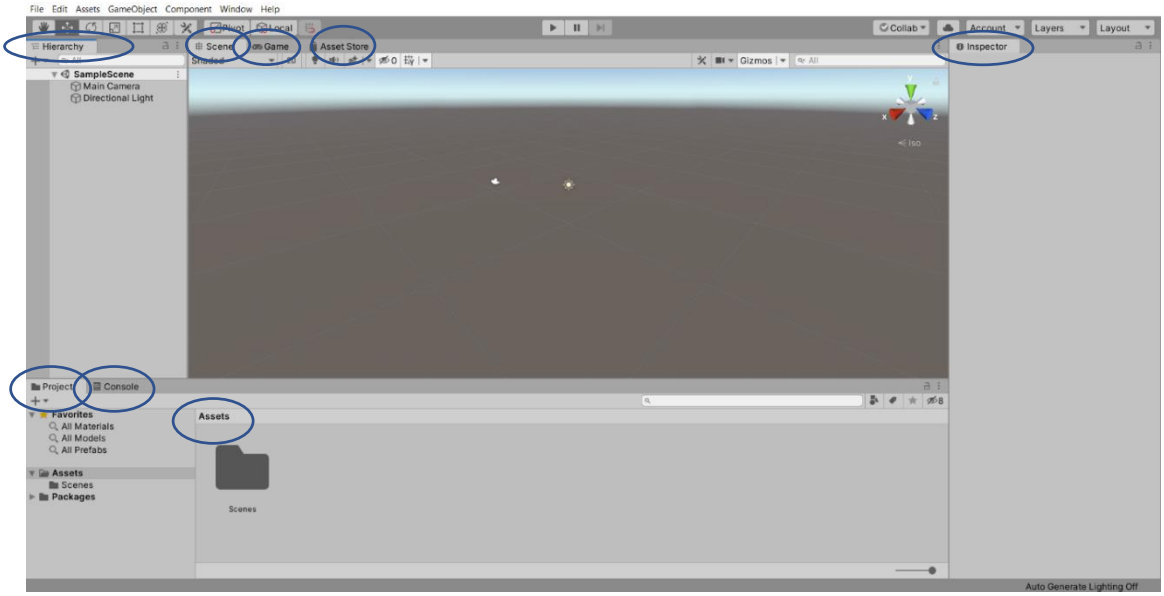



*Figura 66. Ventana principal-Unity*

Dentro de esta vista se puede observar diferentes elementos que deben ser comprendidos para poder llevar a cabo el desarrollo de un proyecto cualquiera en Unity:

**Hierarchy**. La ventana de Hierarchy presenta los elementos que están dentro de la escena, también llamados: **GameObject**. De manera predeterminada, Unity crea la "Main Camera" y el "DirectionalLight". La Main Camera permitirá visualizar la escena una vez iniciada. Por otro lado, el segundo gameobject permite dar luz a la escena. Digamos que simula al "Sol", de manera que, si desplazamos o lo rotamos, cambia la iluminación de la escena.

**Scene**. Representa la ventana donde el usuario puede crear su proyecto a partir de los elementos presentes en la Hierarchy.

**Game**. Esta ventana le permite pre visualizar el proyecto a través de la cámara que exista dentro de la Scene, por defecto la Main Camera.

**Inspector**. La ventana Inspector muestra la información del gameobject que se haya seleccionado. Coordenadas, materiales, propiedades físicas, scripts asociados, etc.

**Console View.** Es la ventana por la cual se muestran los mensajes de compilación del software. Tanto si se producen errores o advertencias como mensajes que el usuario programa para que salgan por pantalla. Resulta bastante útil para el usuario puesto que permite realizar comprobaciones sobre variables o funciones programadas.

**Project**. Visualiza las carpetas de archivos que existen dentro del proyecto, así como los scripts que se hayan creado, los **Prefabs**, materiales etc. Los Prefabs son gameObject almacenados con sus propiedades y componentes que sirven como plantillas reutilizables a partir de las cuales se pueden crear nuevas instancias dentro de la escena. También se muestran los paquetes que se hayan integrado, en este caso habrá que introducir paquetes que permitan la integración del equipo de realidad virtual.

**Assets Store**. Esta es una herramienta muy útil que permite descargar paquetes dentro del proyecto. Tal y como describe Unity en su página web, "El Assets Store de Unity es el hogar de una creciente biblioteca de assets comerciales y gratuitos creados por Unity Technologies y miembros de la comunidad". Aquí se encontrarán otras herramientas útiles para desarrollar cualquier proyecto, pueden incluir: texturas, animaciones, efectos de iluminación y sonido, etc.

### 5.1.2 Componentes principales de un GameObject

Introduzcamos ahora un objeto en la escena. Para ello, en la parte superior, en la pestaña de GameObject, se puede introducir un sólido 3D predeterminado como por ejemplo el cubo que aparece en escena dentro de la figura 56. Una vez introducido, si se selecciona, a través de la ventana Inspector se verán los diferentes componentes que presenta (figura 68).

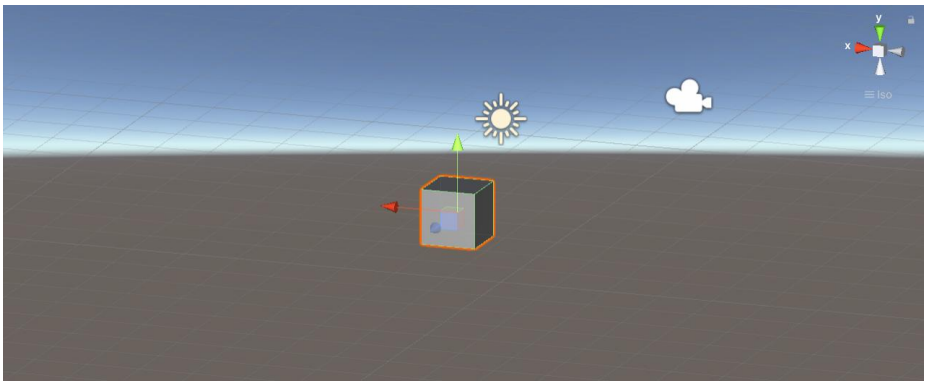


*Figura 67. Cubo gameObject*

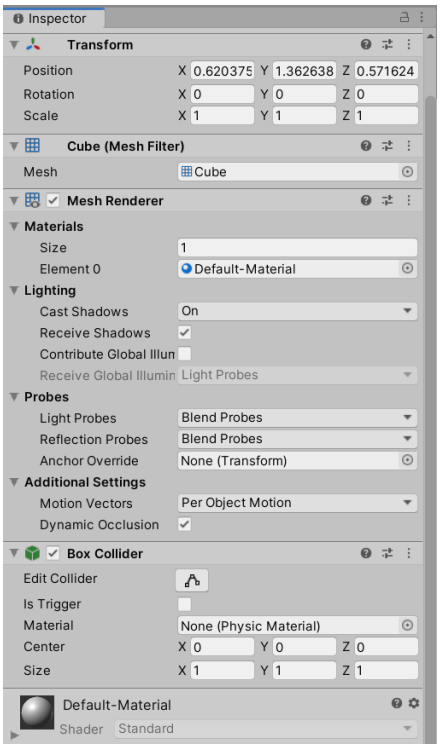



*Figura 68. Inspector*

En primer lugar, la pestaña de **Transform**, muestra las relaciones geométricas principales del objeto: posición, rotación y escala de este.

En la siguiente pestaña, **Mesh Filter** se indica el tipo de malla que da forma al gameobject. La pestaña **Mesh Renderer** coge el tipo de geometría del mesh filter y lo renderiza en la escena en la posición correspondiente según indique la pestaña transform.

Luego se encuentra el **Box Collider**. Este componente consiste en una malla cúbica (de color verde) que recubre nuestro objeto para detectar colisiones en la escena. En caso de que dos gameObject se intercepten, si estos tienen un Collider y uno tiene un componente de **Rigidbody**, el software de simulación física produciría una colisión. La opción Trigger dentro del box Collider desactiva dicha simulación física de colisión, permitiendo que el objeto penetre.

Además del box Collider, existen Colliders de diferente tipo. Según el tipo de malla que se pretenda recubrir: **Sphere Collider** (cuerpos esféricos), **Capsule Collider** (cuerpos cilíndricos) o el tipo **Mesh Collider**. Esta última opción se adapta a la malla de la superficie del gameobject en cuestión, de manera que, a mayor complejidad de forma, mayor cantidad de planos conformaran dicho Collider y por tanto sería más pesado de renderizar.

Finalmente se encuentra el material que presenta el objeto, en este caso un color por defecto que podemos modificar simplemente arrastrando un material creado previamente.

### 5.1.3 Scripts

En cuanto a la programación, en Unity se pueden emplear tres lenguajes diferentes: C#, UnityScript y Boo.

- **C#**

Este lenguaje está orientado a la programación de objetos, sus sintaxis derivan de los lenguajes C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de java. Está diseñado para la infraestructura de lenguaje común, de manera que pueda funcionar en otras plataformas de Microsoft.

- **Javascript**

Es un lenguaje de programación interpretado, destaca por su débil tipado y dinamismo. Todos los navegadores más modernos interpretan este lenguaje en las páginas web, permitiendo mejoras en la interfaz de usuario. Se diseñó con una sintaxis parecida al lenguaje C, aunque adopta nombres convencionales de Java (aun así, no están relacionados).

- **Boo**

Es un lenguaje inspirado en Python, orientado a objetos, de tipos estáticos para la Common Language Infrastructure. Código abierto (licencias tipo MIT/BSD) y se integra perfectamente con Microsoft.NET y Mono.

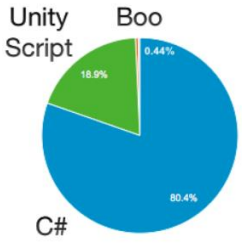



*Figura 69. Lenguajes de programación en Unity [25]*

Como se puede ver, el lenguaje preferido por la comunidad de desarrolladores en Unity utiliza el lenguaje C# para realizar sus proyectos. Del mismo modo este proyecto se desarrollará utilizando la sintaxis de este código.

Los **Scripts** son los archivos en donde el usuario puede desarrollar su código de programación. Estos se asignarán a los gameObjects como un componente cualquiera, de manera que en un mismo gameObject se pueden tener diferentes Scripts.

Para asignar un script a un gameObject debemos desplazarnos hasta la ventana de Inspector y seleccionar la opción *Add Component*.

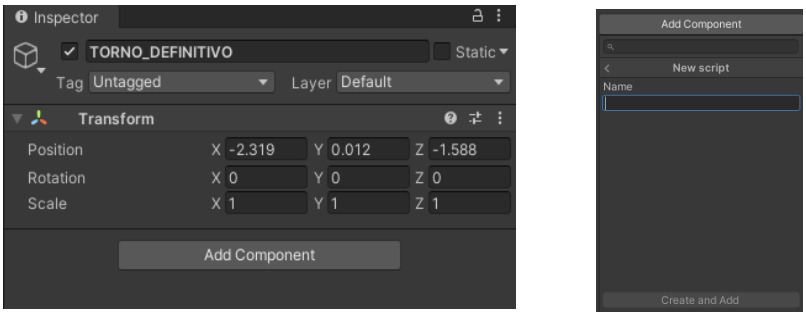



*Figura 70. Crear nuevo Script*

Dentro de esta pestaña se pueden asignar un script ya existente dentro de nuestros Assets o, por otro lado, crear un nuevo script. Todos los scripts presentes en un gameobject

aparecerán junto con los demás componentes de este. Una vez creado, el código se editará utilizando la herramienta de Microsoft Visual Studio.

### *5.1.3.1 Estructura básica de un Script*

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Script_ : MonoBehaviour
6   {
7
8       void Start()
9       {
10
11      }
12
13
14      void Update()
15      {
16
17      }
18  }
19
```

*Script 1. Estructura básica*

En el Script 1 podemos observar la estructura básica que tendría un script sin editar dentro de la aplicación Microsoft Visual. En primer lugar, el código comienza implementando las librerías que se utilizarán dentro del código y que facilitarán el desarrollo del mismo.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
```

*Script 2. Librerías*

A continuación, en la quinta línea del código, se define la "clase", el cuerpo del Script. Cuando se crea un script en Unity lo que se hace es crear una clase, y cuando se asigna este script a uno o varios objetos de nuestra aplicación quiere decir que estamos creando instancias de nuestra clase. Esta recibe el nombre del propio script.

Seguidamente aparece ": MonoBehaviour". Esto es una clase integrada que permite al script "conectar" con el funcionamiento interno de Unity.

Por defecto, el software crea dos funciones: la función *Void Start ()* y *Void Update ()*

Esta función se ejecuta siempre que se arranca la aplicación. Dentro de esta se suelen inicializar variables, así como realizar referencias a otros scripts como ya veremos más adelante. Para hacer referencia a otros gameObject o scripts también se utilizará la función *Void Awake ()*.

La siguiente función que encontramos, la función *Void Update ()*, se ejecuta de manera continua mientras nuestra clase esté activa.

### *5.1.3.2 Declaración de variables*

La declaración de variables dentro del script nos permitirá funcionar con ellas dentro de este. Se pueden crear variables de distinto tipo según el tipo de dato que vayan a almacenar, brevemente, tenemos los siguientes:

**int**. Se utilizan para datos numéricos enteros que a su vez se agrupan en enteros con singo o enteros sin signo. Permiten realizar operaciones aritméticas, de igualdad y comparación, bit a bit. El valor por defecto es “0”.

**Float**. Se utiliza para datos numéricos de punto flotante. Existen tres tipos: float, double y decimal. Este último para cálculos muy precisos. Permiten las mismas operaciones que en el caso de los enteros.

**Char** y **string**. Se utilizan para almacenar datos alfanuméricos. El tipo de datos string nos permitirá almacenar cadenas de caracteres usando comillas dobles.

**Bool**. Mediante este tipo se almacenan datos lógicos (true o false)

| Tipo de datos | Tamaño | Intervalo |
|---|---|---|
| Int | Entero 32 bits con signo | -2147483648 a 2147483647 |
| flloat | 4 bytes | $\pm 1.5x10^{-45}$ a $\pm 3.4x10^{38}$ |
| string | 2 bytes por carácter | Secuencia de caracteres |
| bool | 1 bit | True o false |

*Tabla 4. Tipos de datos*

A parte de los tipos de datos comunes, este lenguaje está referido a objetos, de manera que dentro de estos scripts también se podrán crear variables que hagan referencias a otros gameObject y así leer o modificar el estado de alguno de sus componentes. Esto se verá más adelante junto con la programación en sí misma de nuestro modelo de torno.

En función del lugar donde se declaren dentro de nuestro script, las variables pueden ser: variables locales y variables de clase.

**Variables locales**. Estas variables se declaran dentro de una función o método. Por ejemplo:

```
public void SUMA(int valor_1, int valor_2)
{
    int valor_sumar = valor_1 + valor_2;
}
```

*Script 3. Función SUMA*

En este caso se ha definido la función *SUMA ()* que recibe dos valores enteros. Dentro de esta función se ha declarado la variable local *valor_sumar* que almacena la suma de esos valores. Esta variable solo se utiliza dentro de esta función, de manera que no se puede acceder a ella fuera de esta función.

**Variables de clase**. Las variables de clase se declaran al inicio de un script, debajo de la clase y pueden ser leídas o modificadas por todo el script.

```
public class Script_ : MonoBehaviour
{
    int valor_suma;

    public void SUMA(int valor_1, int valor_2)
    {
        valor_suma = valor_1 + valor_2;
    }
}
```

*Script 4. Variables de clase*

Finalmente, las variables pueden ser definidas como públicas o privadas. Como bien podemos entender, las variables públicas pueden ser accedidas a través de otros scripts, incluso a través del Inspector donde el usuario puede modificar su valor. Sin embargo, las privadas solo se utilizarán durante la ejecución del script en concreto.

```
public class Script_ : MonoBehaviour
{
    private int valor_suma;
    public int sumando1;
    public int sumando2;

    void Start()
    {
        valor_suma = sumando1 + sumando2;
        Debug.Log("suma: " + valor_suma);
    }
```

*Script 5. Variables públicas y privadas*

Este código nos va a permitir introducir dos valores. Posteriormente, cuando arranquemos el programa se ejecutará la función *Void Start* que realizará la suma de ambos valores mostrando el resultado a través de la ventana Console.

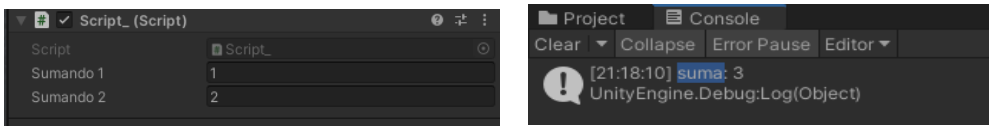



*Figura 71. Resultado de la función SUMA*

## 5.2 Exportación del modelo

Unity acepta una serie de formatos a la hora de importar modelos. En nuestro caso, al haber generado el torno mediante SolidWorks, tendremos que hacer una serie de pasos para convertir el documento en el formato adecuado.

Unity acepta dos tipos de archivos diferentes a la hora de importar un modelo: Formatos de exportación 3D (FBX u OBJ) o formatos propios de aplicaciones 3D (Studio Max o Maya).

En nuestro caso, al haber modelado el torno a través del software de SolidWorks, necesitaremos realizar varios pasos a la hora de exportar el modelo en alguno de esos formatos.

**Formatos de exportación 3D (.FBX, .DAE, .3DS, .DXF y .OBJ)**

| VENTAJAS | DESVENTAJAS |
|---|---|
| ▪ Solo se exportan datos útiles.<br>▪ Son archivos de pequeño tamaño.<br>▪ Soporta otros paquetes 3d para cuyo formato propietario no tenemos soporte directo.<br>▪ Tiene un enfoque modular. | Tenemos que realizar una exportación doble de manera que podemos perder archivos en el transcurso. |

*Tabla 5. Ventajas y desventajas de formatos de exportación 3D*

**Formatos propios de aplicaciones 3D (Max, Maya, Cinema4D, Lightwave, etc)**

| VENTAJAS | DESVENTAJAS |
|---|---|
| ▪ Proceso rápido<br>▪ Mayor simplicidad | ▪ Los archivos pueden contener información no necesaria<br>▪ Los archivos grandes pueden ralentizar Unity |

*Tabla 6. Ventajas y desventajas de formatos propios de aplicaciones 3D*

En nuestro caso se va a utilizar una nueva herramienta denominada Pixyz, aunque para algunos elementos puntuales se trabajará con Blender que permitirá realizar la exportación a .FBX.

### 5.2.1 Pixyz

Pixyz es una nueva herramienta creada para importar documentación de ingeniería 3D a softwares de videojuegos como Unity. De esta manera nos permite importar cualquier archivo de CAD nativo (CATPart, SolidWorks, JT, STEP, etc.).

Pixyz se integra fácilmente dentro del editor de Unity a través de la descarga e instalación de Pixyz plugin. Para ello se debe acceder a la página web de Pixyz y descargar el archivo *Pixyz Plugin for Unity*.

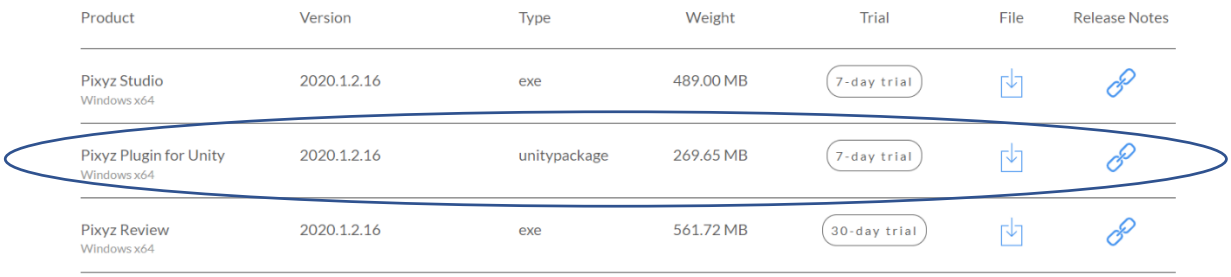



*Figura 72.Descarga Pixyz plugin*
*Fuente: Download - Pixyz Software (pixyz-software.com)*

Una vez hecha la descarga se tendrá que iniciar la aplicación de Unity para su instalación dentro del proyecto. Esto se hará a través de la importación de paquetes a la cual podemos acceder mediante la pestaña Assets que se encuentra en la parte superior de la ventana.

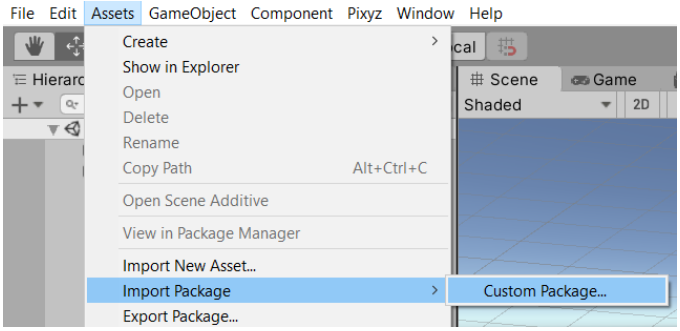



*Figura 73. Import Package*

Se busca el archivo que hemos descargado anteriormente y una vez instalado encontraremos una nueva pestaña en la parte superior con el nombre de la herramienta. Si bien, esta herramienta que se acaba de instalar no es gratuita, tiene un coste medio de 1800€/años, de manera que se utilizará la prueba gratuita de 7 días que ofrece la aplicación para poder importar nuestro modelo.

Una vez conseguida la licencia gratuita se procederá a importar el modelo. Para ello se despliega la pestaña de Pixyz y se selecciona la opción de *Import Model*.

### 5.2.2 Blender

El proceso de importación desde Blender es algo más tedioso de manera que solo ha sido utilizado para algunos archivos que se modificaron después de que expirase la prueba gratuita de Pixyz.

Se dice que es un proceso más tedioso ya que se debe realizar una doble importación. En primer lugar, una vez creado el modelo en SolidWorks se tendría que exportar el modelo en formato .STL. Este es un formato de archivo CAD que permite definir la geometría de objetos 3D, pero excluye información como color, texturas, etc. Una vez exportado a este formato, desde la aplicación de Blender se puede abrir el documento y exportarlo en formato .FBX.

Este software de edición 3D permite realizar otras muchas funciones, sin embargo, solo se utilizará como transición a la hora de importar nuestro modelo a Unity.

Una vez posicionado el modelo, debido a la exportación desde SolidWorks, puede haber modificaciones en el sistema de referencia y en el origen de coordenadas que se deban corregir.

A la hora de realizar una comparativa entre ambos métodos de exportación, Pixyz resulta la mejor opción por dos aspectos que podemos destacar frente a la importación desde Blender:

**Suavizado de superficies curvas**. Esto se debe a que cuando se importa un modelo en formato STL, las superficies curvas presentan el efecto “flatten surface” por el que se perciben los planos que forman esa superficie. Esto se puede corregir desde Blender mediante el suavizado de caras, cosa que Pixyz ya hace por defecto.

*Figura 74.Calidad exportación Pixyz/Blender*

**Respeto de la Jerarquía en el modelo**. Cuando se exporta el modelo desde Pixyz, este no solo guarda información acerca de texturas o color dentro del modelo, además respeta la jerarquía dentro de los diferentes ensamblajes que se hayan incorporado. De esta manera se puede acceder a un ensamblaje y desplegarlo para seleccionar las diferentes piezas que lo componen (al igual que en SolidWorks).

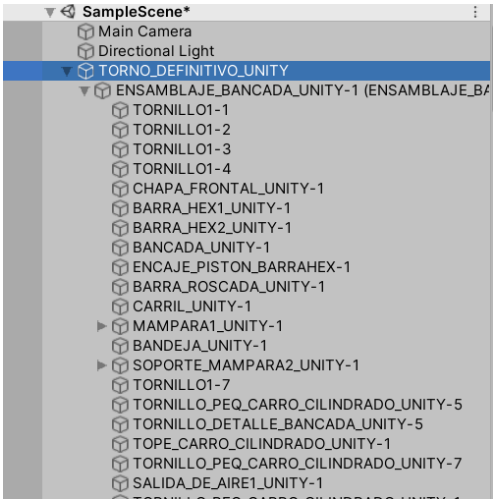



*Figura 75. Jerarquía modelo Pixyz*

## 5.3 Configuración de Unity para soporte en VR

Unity no es un software exclusivo para realidad virtual, por tanto, se deben realizar una serie de configuraciones que permitan integrar el equipo de realidad virtual. En este apartado se desarrollará el proceso de instalación del equipo Oculus, guiándonos a través de los tutoriales que ofrecen los equipos de ingeniería de software de ambas compañías [26].

La primera configuración que se debe hacer en el proyecto es habilitar el soporte de realidad virtual dentro de este, para ello abrimos *Project Settings* y habilitamos la opción de *XR Plug-in Management*, y seleccionamos la opción de *Oculus*.

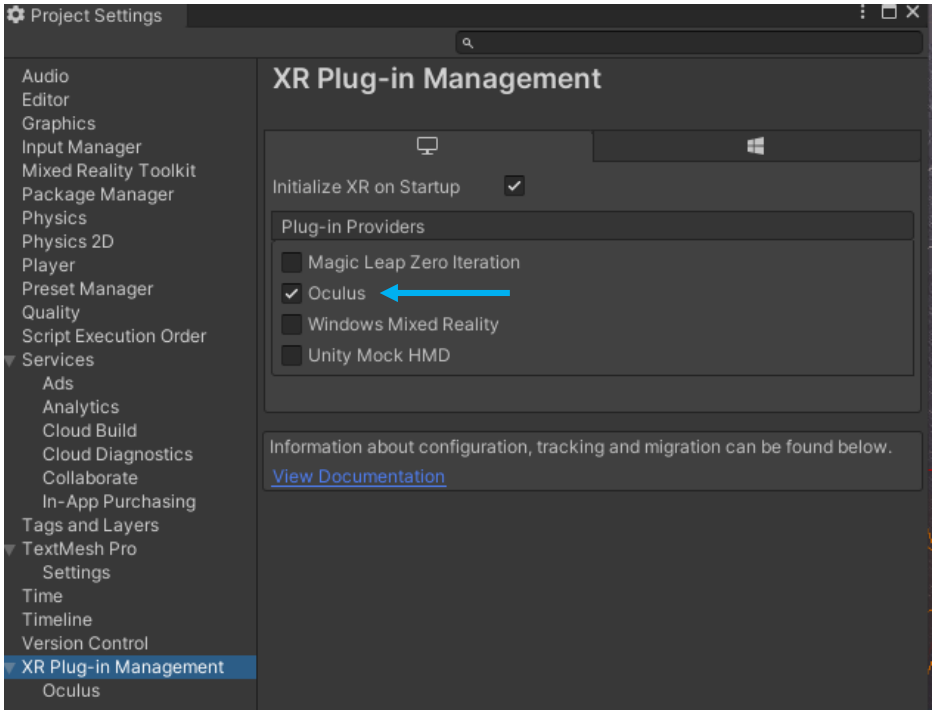



*Figura 76. XR Plug-in Management*

### 5.3.1 Paquetes integrados

*5.3.1.1 Oculus Integration*

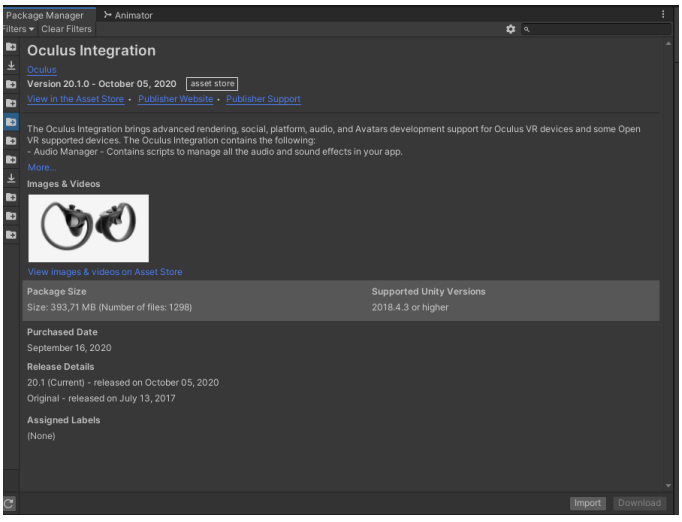



*Figura 77. Oculus Integration*

Unity proporciona soporte de realidad virtual integrado para dispositivos Oculus mediante el paquete de integración de Oculus. Este agrega scripts, Prefabs, muestras y otros recursos para complementar la compatibilidad integrada de Unity. El paquete incluye una interfaz para controlar el comportamiento de la cámara VR, un Prefab de control en primera persona, una API de entrada unificada para controladores, características de representación, herramientas de depuración y mucho más que iremos viendo a continuación.

Para su instalación debemos ir al *Assets Store* y descargarlo. Una vez hecho, desde el *package manager* podremos importarlo en la escena.

Una vez terminado el proceso de instalación, en la ventana inferior de Project nos aparecerá una nueva carpeta, dentro de Assets, denominada *Oculus*, con todo el contenido que nos ofrece esta herramienta para desarrollar nuestra aplicación en realidad virtual. Dentro del contenido que ofrece esta herramienta nos centraremos en la carpeta de VR, si la desplegamos encontraremos los Prefabs. Esta carpeta incluye:

**OVRCameraRig**. Una cámara VR personalizada que optimiza el renderizado para una pantalla estereoscópica en el dispositivo Oculus. Proporciona acceso a OVRManager, que es una interfaz al hardware VR.

**OVRPlayerController**. Permite al jugador moverse en el entorno vitual. Incluye componentes y objetos secundarios necesarios para el control 3D. Incluye prefab de OVRCameraRig del que acabamos de hablar.

**OVRCubemapCaptureProbe**. Captura una imagen de pantalla estática 360 de la aplicación desde la perspectiva de la cámara de escena, mientras se ejecuta la aplicación. Le permite capturar la imagen de pantalla en un momento específico después del lanzamiento, en una pulsación de tecla específica o con el uso de la función estática *OVRCubemapCapture.TriggerCubemapCapture*.

**OVRHandPrefab**: Implementa las manos como asequibilidad de entrada.

De estos cuatro prefabs solo utilizaremos el OVRCameraRig. Este prefabs sirve de elemento sustituto a la Main Camera. De manera que cuando estemos dentro de un aplicación de realidad virtual debemos eliminar esta última.

Introduciendo el prefab en la escena, en la pestaña de jerarquía podemos ver la composición de este prefab. El prefab OVRCameraRig contiene el *Tracking Space* para ajustar la relación entre el marco de referencia de seguimiento de cabeza y su mundo. Debajo de este, se encuentra el *CenterEyerAnchor* , que es la cámara principal de Unity, dos objetos de juego de anclaje para cada ojo, *LeftEyeAnchos* y *RightEyeAncho*, y anclajes a la izquierda y a la derecha para los controladores, *LeftHandAnchor y RightHandAnchor*.

Cuando habilitamos el soporte de Realidad Virtual en Unity, el dispositivo montado en la cabeza pasa automáticamente la referencia de seguimiento de cabeza y posicional a Unity. Esto permite que la posición y la orientación de la cámara coincidan finamente con la posición y la orientación del usuario en el mundo real. Los valores de pose con seguimiento directo anulan los valores de transformación de la cámara, lo que significa que la cámara siempre está en una posición relativa al objeto del reproductor.

Hay dos scripts principales asociados al prefab OVRCameraRig: *OVRCameraRig.cs* y *OVRManager.cs*. Cada script proporciona ajustes para la cámara, la pantalla, el seguimiento, la calidad y el rendimiento de la aplicación que podemos personalizar en la ventana de Inspector.

#### *5.3.1.2 VRTK*

A continuación, integraremos otro paquete de realidad virtual: el paquete VRTK. Este incorpora también una colección de soluciones útiles y reutilizables a problemas comunes que se encuentran al construir una aplicación de realidad virtual.

VRTK tiene como objetivo ayudar a la productividad acelerando el proceso de creación, desde la creación de prototipos de ideas hasta la creación de soluciones completas, como es nuestro caso. Para su instalación debemos irnos a la carpeta de *Packages* en el archivo de nuestro proyecto y abrir el archivo de texto *manifest. json* para introducir el siguiente 0

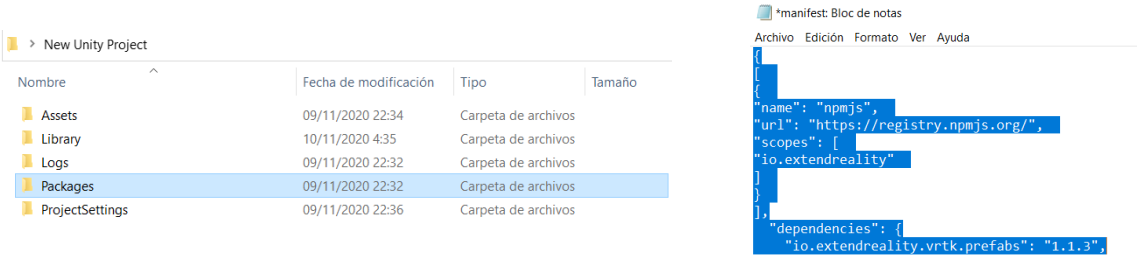



*Figura 78. Instalación VRTK*

Una vez realizada dicha operación, se incorporarán a nuestra ventana de proyecto, las diferentes herramientas que utilizaremos en función de los requerimientos de nuestra aplicación. Entre los prefabs que vamos a utilizar y que desarrollaremos más adelante, podemos mencionar los siguientes:

- *Rotational Joint Drive*
- *Directional Joint Drive*
- *Interactable.Primary_Grab.Secondary_Swap*
- *Interactor*
- *Tracked Alias*

### 5.3.2 Configuración de la cámara de realidad virtual y presencia de manos.

- **Paso 1**

Lo primero que debemos hacer es introducir en nuestra escena los Prefabs de OVRCameraRig y TrackedAlias. Para ello buscamos en la ventana de Project, en las carpetas que hemos importado anteriormente, y lo arrastramos a nuestra escena.

- **Paso 2**

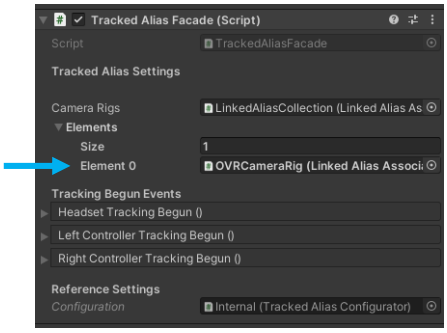
Una vez abiertos los Prefabs en la jerarquía, debemos establecer la asociación entre ambos componentes. Seleccionando el OVRCameraRig, en el inspector introducimos el script *LinkedAliasAssociationCollection* y a continuación agregamos las siguientes entradas adecuadamente:

— *TrackingSpace* a *Play Area*
— *CenterEyeAnchor* a *Headset*
— *CenterEyeAnchor* a *Headset Camera*
— *LeftHandAnchor* a *Left Controller*
— *RightHandAnchor* a *Right Controller*

- **Paso 3**

Para completar la asociación de ambos Prefabs, seleccionamos TrackedAlias en la ventana de Hierarchy. Ahora en el Inspector, buscamos el script asociado

*TrackedAliasFacade* y en la pestaña de *Elements* cambiamos *Size* a 1. Por último, solo debemos arrastrar el Prefab de OVRCameraRig de la Hierarchy hasta *Element 0.*




*Script 6. Tracked Alias Facade*

Completados estos pasos, el usuario ya sería capaz de arrancar la aplicación, y tener una visión espacial dentro de la escena. Para completar esta etapa vamos a pasar a introducir la presencia de manos dentro de la escena.

- **Paso 4**

Para introducir la presencia de manos dentro de la escena, el paquete de Oculus introduce un Prefab para cada una: *CustomHandLeft* y *CustomHandRight*. Estos Prefab contienen dos scripts asociados, el script *Hand* se encargará de la animación de la mano dentro de la escena a través de las entradas de los controladores. Una vez seleccionados, los arrastramos dentro del Prefab TrackedAlias.

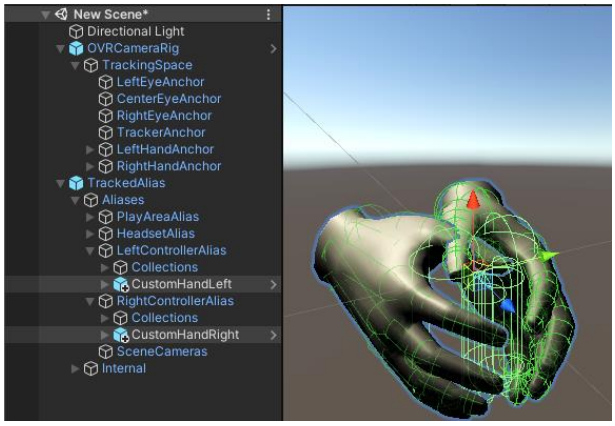



*Figura 79. Presencia de manos en la escena*

### 5.3.3 Movimiento dentro de la escena

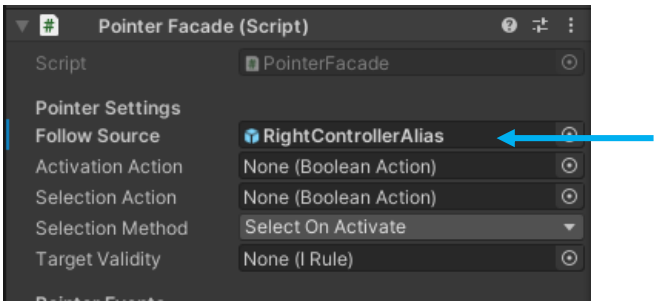
Para movernos es la escena se ha optado por un sistema de Teletransporte mediante el cual el jugador puede desplazarse dentro de la escena a través de los controladores OculusTouch.

- **Paso 1**

En la ventana de Project buscamos el Prefab *Objectpointer.Curved* y lo introducimos dentro de nuestra escena. En el caso de que puedan existir duplicados podemos cambiar

el nombre al Prefab. Finalmente, este Prefab se llamará *TeleportCurved.R* haciendo referencia al puntero de Teletransporte de la mano derecha.

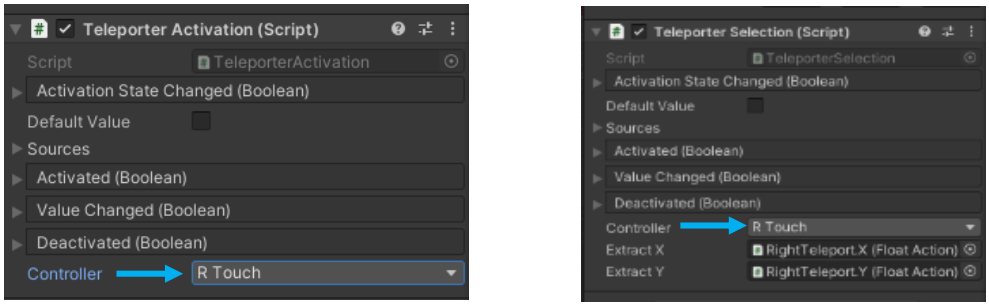
Dentro de este Prefab encontramos el script *PointerFacade* mediante el cual podemos asociar el puntero en este caso, a la mano derecha. De esta manera, seleccionando el Prefab, en *Follow Source* arrastramos el controlador de la mano derecha de TrackedAlias.




*Script 7. Pointer Facade*

- **Paso 2**

Creamos los gameObjects *TeleporterSelection.R* y *TeleporterActivation.R* y agregamos los scripts *TeleporterSelection* y *TeleporterActivation* de manera correspondiente. Seleccionando TeleporterActivation.R en la hierarchy debemos seleccionar el tipo de controlador dentro del script. En este caso hacemos referencia a: *RTouch.*




*Script 8. Teleporter Activation y Teleporter Selection*

Dentro de *TeleporterSelection.R* creamos dos gameObjects hijos: *RightTeleport.X* y *RightTeleport.Y* y les añadimos el script *FloatAction*. En el tipo de controlador seleccionamos el mismo que anteriormente y en las opciones de *Extract* introducimos los hijos según la componente asociada.

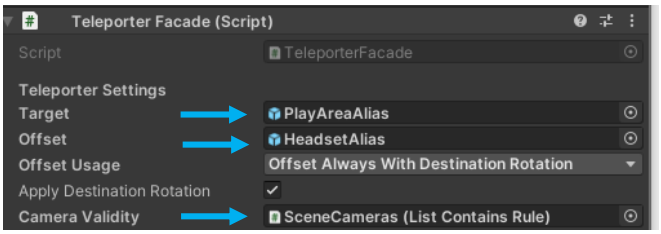
Por último, habría que asociar los prefab *TeleporterSelection.R* y *TeleporterActivation.R* dentro del script *pointer facade*.

- **Paso 3**

Ahora añadiremos el prefab *Teleporter.Instant* en nuestra escena. Una vez en Hierarchy lo seleccionamos y abrimos el script asociado *TeleporterFacade* en Inspector. Dentro de este encontramos la opción *Target.* Esta hace referencia a que queremos mover cuando se produce el "Teletransporte", en nuestro caso, esto es igual a que se desplace el área de juego *(PlayAreaAlias).*

La opción *Offset* sirve para mantener la orientación de la cámara cuando se produce la acción, por lo que debemos arrastrar el componente referido a la posición de nuestra cabeza (*HeadsetAlias*).

Finalmente, arrastramos nuestra *SceneCamera* dentro de *camera validity*.




*Script 9. Teleporter Facade*

- **Paso 4**

Para introducir los puntos donde el usuario pueda "teletransportarse" dentro de la escena, debemos definir una *Layer* o capa. Para ello, buscamos *Layers* en el inspector y la editamos. En este caso hemos llamado a esta nueva capa: *Teleportable*.

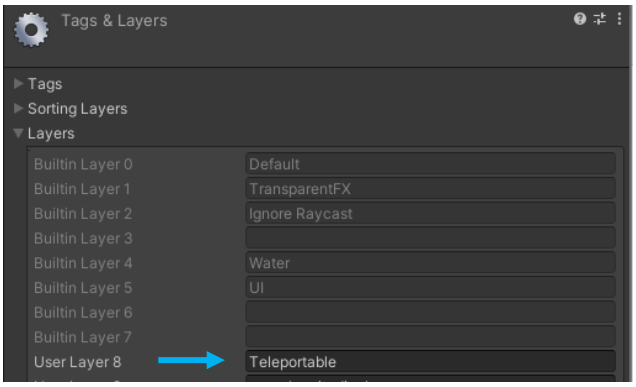
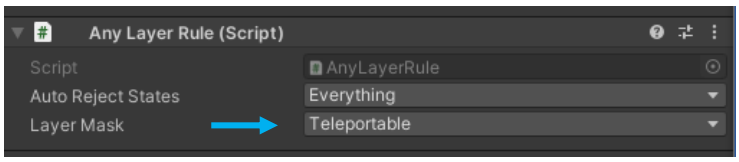



*Figura 80. Layer*

Una vez creada debemos asociar esa capa al script *AnyLayerRuler*. De esta manera creamos un nuevo gameobject al que llamamos *teleportationRule* y le asociamos este script. Dentro de este script solo tenemos que seleccionar nuestra capa en la opción *Layer Mask*.




*Script 10. Any Layer Rule*

Finalmente, si volvemos al *Teleporter.Instant* y abrimos el script de *TeleporterFacade*, encontramos la opción de *Target Validity* donde debemos arrastrar el gameobject que hemos creado ahora asociado con la nueva *Layer*. Así pues, conseguimos que el usuario pueda solo desplazarse a aquellos destinos cuya *Layer* sea de este tipo. Lo mismo debemos hacer con el componente *TeleportCurved.R*. Abrimos el script asociado y procedemos de la misma manera.

- **Paso 5**

Para que el desplazamiento se lleve a cabo debemos asociar los Prefabs *TeleportedCurved.R* y *Teleporter.Instant*. Queremos que cuando el usuario ejecute la acción de teletranporte a través del joystick derecho, el sistema lo lea y realice dicha acción. Para ello *TeleportedCurved.R*, que es quien recibe la activación por parte del usuario, debe mandar la información a *Teleporter.Instant* para que se produzca dicho desplazamiento. Esto se consigue mediante la herramienta de UnityEvent.

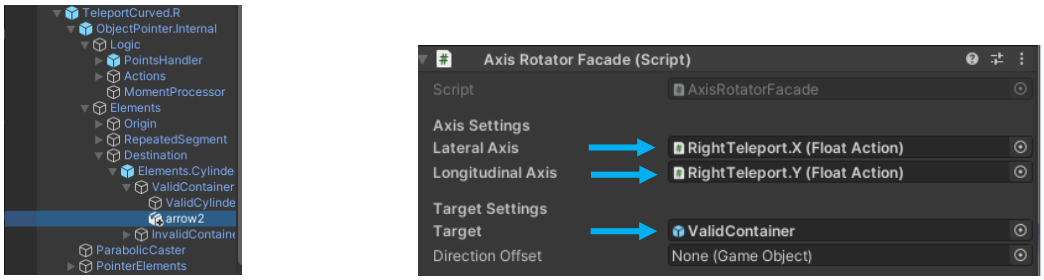
Cuando *Selected* esté validado, activará la función *Teleport* dentro del script *TeleportFacade* asociado al gameobject *Teleport.Instant* y se producirá el Teletransporte.

- **Paso 6**

Cuando se produce la acción de Teletransporte, el usuario debe poder introducir a través del joystick, el sentido de giro, de manera que cuando termine la acción, la orientación dentro de escena sea la deseada. Para ello debemos introducir el *AxisRotator*, denominado *AxisRotator.R* en nuestra escena.

Dentro del *AxisRotator* lo que hacemos es leer de nuevo los valores de desplazamiento del Joystick a través de los componentes *RightTeleport.X* y *RightTeleport.Y*.

Para visualizar la dirección introduciremos un indicador dentro del puntero de Teletransporte que podremos mover a través de este script. En este caso, el indicador se trata de una flecha. De esta manera, cada vez que el usuario desplace el joystick podrá visualizar el punto de desplazamiento, así como la orientación de la cámara. Por tanto, si queremos asociar el movimiento del AxisRotator a nuestro indicador, debemos arrastrar el padre del gameobject de la flecha a este script




*Script 11.Axis Rotator Facde*

### 5.3.4 Interacción con los objetos de la escena

A continuación, realizaremos la configuración necesaria para poder interactuar con los objetos presentes en la escena. Para el sistema de agarre, el paquete de Oculus Integration incorpora los scripts *OVR Grabbable* y *OVRGrabber* mediante los cuales podemos agarrar objetos. Sin embargo, desactivaremos estos scripts de los Prefabs de las manos que hemos incorporado anteriormente *(CustomHandRight* y *CustomHandLeft*) y utilizaremos el sistema de agarre de VRTK. Para ello debemos seguir una serie de pasos que nos permitan relacionar la API de OculusTouch con este nuevo sistema de agarre.

- **Paso 1**

Para empezar con la configuración buscaremos el Prefab *Interactor* en nuestros Assets, y lo arrastraremos hasta el Prefab *Tracked Alias*, dentro de uno de nuestros controladores junto con una de nuestras manos.

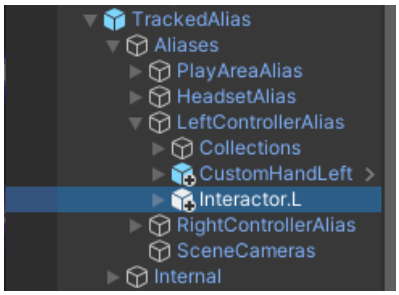
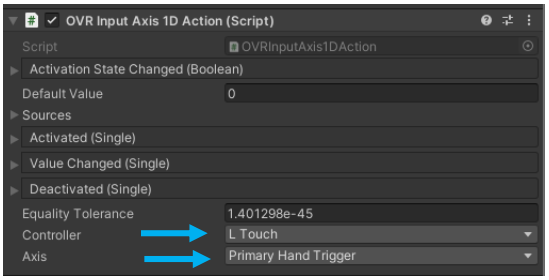



*Figura 81. Tracked Alias*

- **Paso 2**

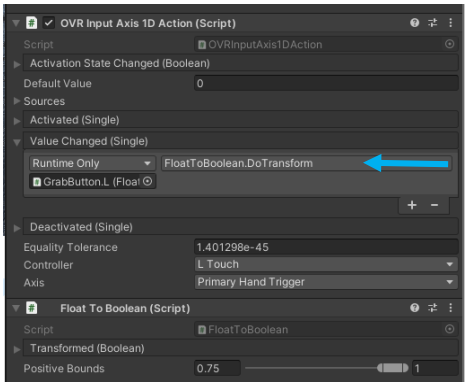
Ahora crearemos un gameObject que llamaremos *GrabButton.L*. Este componente nos servirá para leer cuando el usuario lleve a cabo la acción de agarrar. Para ello, dentro de inspector, buscaremos el script *OVRInputAxis1DAction*, este servirá para obtener la entrada de cualquier botón de nuestro controlador. En este caso queremos leer el botón de agarre del controlador, este es: *Primary Hand Trigger*.

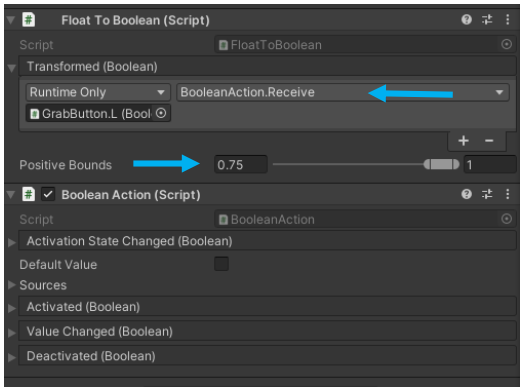



*Script 12. OVR Input Hand Trigger*

Esta entrada la leemos con una variable de tipo float, sin embargo, nosotros solo queremos saber si se produce o no el agarre, de manera que convertiremos esa entrada float en otra variable tipo bool (true o false). Para ello debemos introducir el script *Float to Boolean* y asociar ambos scripts.

En *Positive Bounds* especificamos cuanto debemos pulsar el botón para que la acción se considere agarre. En caso de que el valor float que recibe del primer script superase ese valor, entonces la variable de salida sería "true". En caso contrario, la variable de salida sería false. Para recibir la variable tipo bool de salida necesitamos a su vez otro script, *Boolean Action*.

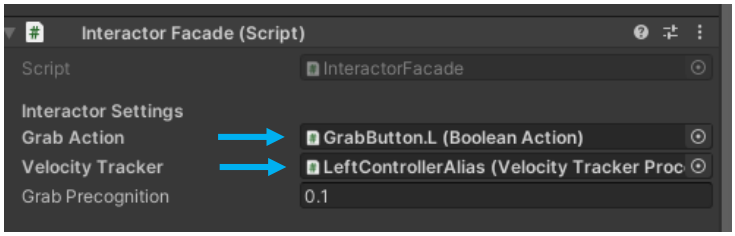



*Script 13. Lectura del desplazamiento del botón Hand Trigger*

- **Paso 3**

Realizados los pasos anteriores, debemos asociar el *GrabButton.L* con el *Interactor*. De esta manera el script *Interactor Facade* asociado al *Interactor* recibirá una entrada de valor "true" en caso de que se produzca el agarre y "false" en caso contrario.




*Script 14. Interactor Facade*

Este script lleva a cabo la acción de agarrar, para ello básicamente este script activa la opción *IsKinematic* del componente Rigidbody del objeto agarrado, de manera que ahora obliga a que este siga el mismo desplazamiento que la propia mano modificando su *Transform*.

Una vez que hemos configurado el sistema de agarre en ambas manos repitiendo el proceso para la mano izquierda, debemos configurar aquellos objetos que el usuario puede agarrar. Para ello se introduce: *Interactable.Primary_Grab.Secondary_Swap*. Para poder interactuar con cualquier gameObject, basta con introducirlo del Prefab anterior.

## 5.4 Programación del Torno en VR

A continuación, abordaremos el proceso de programación de los diferentes componentes que conforman nuestra máquina y que permiten el funcionamiento de la misma. No trataremos de explicar el funcionamiento de cada uno de los scripts, sino aquellos que sean necesarios para poder explicar las diferentes funciones que hemos llevado a cabo, así como códigos y otros conceptos que veremos a continuación.

Antes de abordar la programación de los diferentes componentes del torno explicaremos brevemente los Prefabs de interacción que nos brinda VRTK para mejorar la experiencia de realidad virtual.

- **Rotational Joint Drive**

Este Prefab se utilizará para la mayoría de elementos en los que necesitemos interactuar con nuestras manos virtuales para efectuar movimientos giratorios. Se utilizará en los mandos de selección de la bancada, las manivelas del contrapunto o en la mampara protectora.

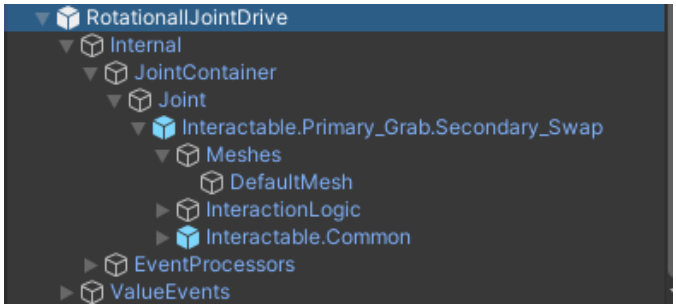



*Figura 82. Rotational Joint Drive*

Si introducimos este Prefab en nuestra escena, encontramos una serie de desplegables dentro de este. Empezaremos explicando el elemento más básico y del que ya hemos hablado anteriormente: *Interactable.Primary_Grab.Secondary_Swap*. Este Prefab permite la comunicación con el *Interactor* a través del *Interactable facade*, de manera que cualquier objeto que esté presente dentro de este Prefab puede ser agarrado. Para ello debemos colocar nuestro objeto dentro de la pestaña *Meshes*. Por defecto encontraremos el *Default Mesh* (cubo) pero este puede ser sustituido por cualquier otro objeto.

Para poder simular la rotación del gameobject se hace uso del componente *Joint* o articuluación. Para poder simular una articulación en Unity, debemos introducir dos componentes principales: **Rigidbody** y cualquier componente articulado como **Hinge Joint** que se utilizará para mover el gameObject alrededor de un eje de giro.

El componente Rigidbody va a estar presente en muchos de los componentes de nuestro proyecto. Esto permite a cualquier objeto actuar bajo la simulación física de Unity, de manera que cualquier gameObject esté expuesto a efectos de recibir fuerzas externas y torque. De esta forma tenemos la opción de activar *Gravity* para que el objeto este influenciado por la gravedad o en su caso, activar la opción *Is Kinematic*, mediane la cual el objeto no se mueve por simulación física sino solo por su *Transform*. A modo de ejemplo, cuando interactuamos con un objeto, el *Interactor facade* se encarga de activar esta opción para tener el control de este como hemos explicado anteriormente.

En cuanto al componente Hinge Joint, este es el responsable del giro del elemento. Está fijado a su Gameobject padre *JointContainer*, que también debe presentar un rigidbody. Si nos fijamos en la ilustración, en la opción de *Axis* , vemos como el eje “z” es el único que no esta a “0”, por lo que el elemento que hayamos introducido dentro del prefab, en este caso el *Default Mesh*, girará entorno a su eje “z”

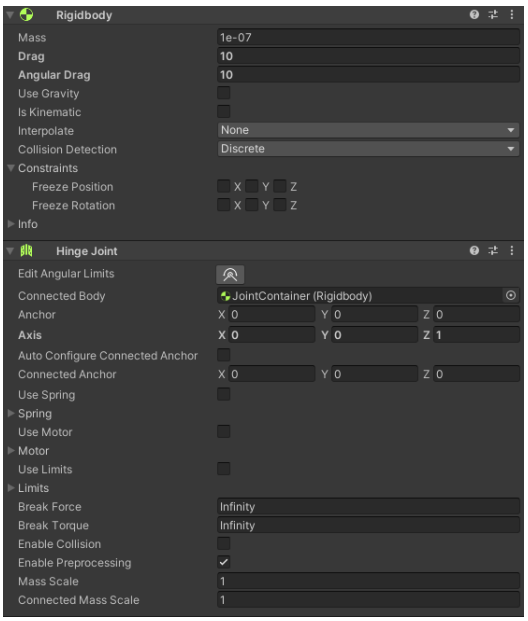



*Figura 83. Hinge Joint*

Por último, si seleccionamos el padre de este prefab y nos vamos a la ventana de Inspector podemos ver varios eventos así como configuraciones que podemos hacer a la hora de interactuar con el objeto. La opción *Drive Axis* nos permitirá seleccionar el eje de giro, dependiendo del eje seleccionado el componete Hinge Joint modificará su eje de revolución. Por último también podemos establecer el rango de giro con la opción *Drive Limit*

- **Directional Joint Drive**

Este Prefab tiene un esquema similar al anterior, aunque en este caso el objeto no se agarra, solamente se desplaza en una dirección. Este tipo de Prefabs se utilizará en los distintos pulsadores o botones que presenta nuestro torno.

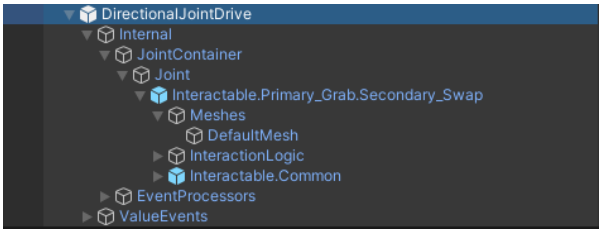



*Figura 84. Directional Joint Drive*

A diferencia del Prefab anterior, el gameObject *Joint* ya no presentará una componente Hinge Joint, ya que esta es específica para giros, sino que incorpora el componente **Configurable Joint**. Este tipo de articulaciones por su parte, son totalmente personalizables, como vemos en el configurador del Inspector.

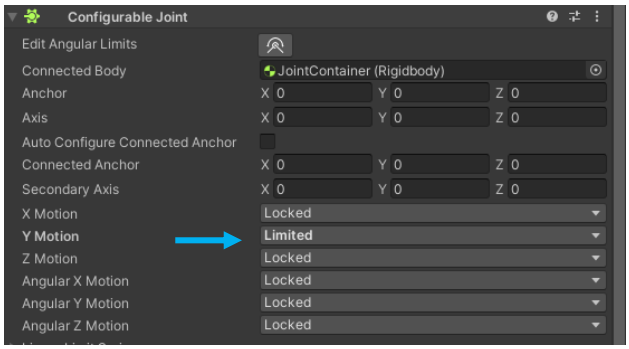



*Figura 85. Configurable Joint*

En este Prefab solo se permite el movimiento en una dirección, de manera que tendremos restringidos cualquier giro o desplazamiento que no sea en la dirección que hemos seleccionado dentro del *Directional Drive Facade*, en el padre del Prefab. Además, también podemos limitar, del mismo modo que en el caso anterior, el desplazamiento de nuestro objeto a través de la opción *Drive Limit*.

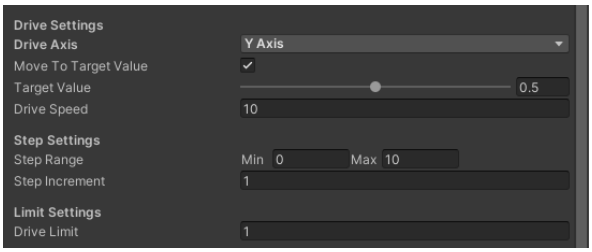



*Figura 86. Configurador del Joint Drive*

Si queremos introducir este Prefab para un pulsador, debemos introducir algunos scripts de los que ya hemos hablado: *Float to Boolean*, *Boolean Action* y *Physical Button Pressed*. Lo que hacemos es leer el desplazamiento de nuestro objeto dentro del límite de desplazamiento que hemos seleccionado en la configuración anterior. Este valor lo lee el script *Float to Boolean* que establece un desplazamiento de 0.8 para ser considerada como tal. En caso de ser así, el script *Boolean Action* recibe el valor de salida en una variable

tipo bool que estará a “true” en caso de ser una pulsación. Esto lo recoge el script *Physical Button Pressed* a través de la función *SetState()* que activa el evento *onPressed* .

### 5.4.1 Husillo

Vamos a comenzar desarrollando la programación de uno de los elementos principales dentro del torno, así como uno de los más complejos debido a la cantidad de elementos que involucra.

#### *5.4.1.1 Velocidades del Husillo*

| 85 | 140 | 200 | 320 | 580 | 850 | 1400 | 2000 |
|---|---|---|---|---|---|---|---|
| A | B | C | D | A | B | C | D |

```
private int[,] velocidad_plato = new int[2, 4] { { 85, 140, 200, 320 }, { 580, 850, 1400, 2000 } };
```

*Script 15. Declaración de la matriz de velocidades*

Lo que haremos será crear una variable de tipo Array, en este caso una matriz [2,4] en la que almacenemos los datos de velocidades en rpm. Esta matriz se definirá dentro de un script que asociaremos al Husillo, y al cual accederemos a través de los mandos de selección de velocidades. Estos mandos enviarán una posición de fila y columna dentro de la matriz de velocidades en función de la posición que esté marcada en cada momento.

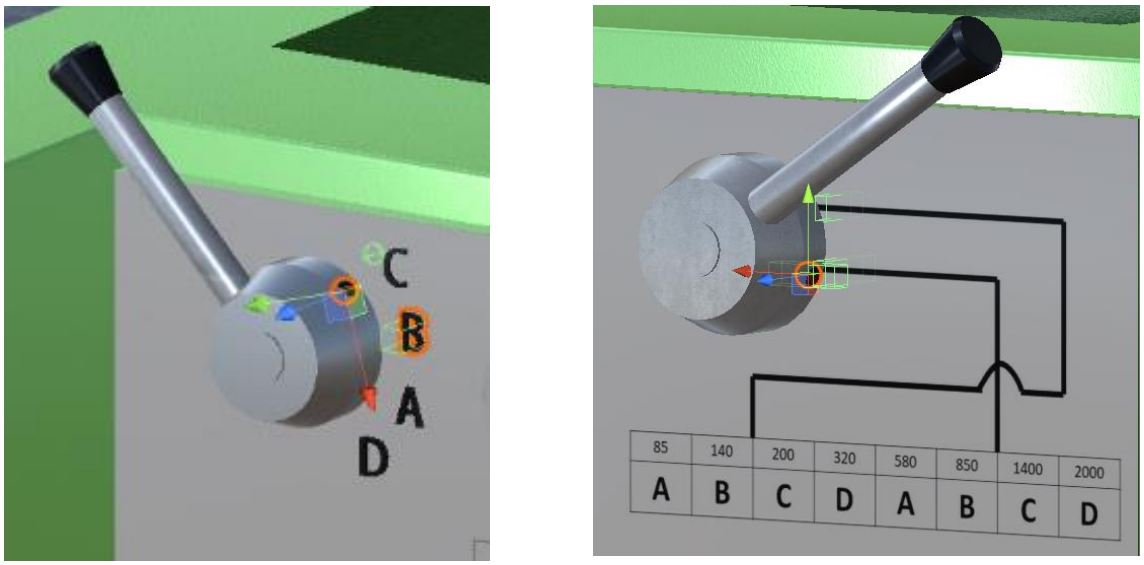



*Figura 87. Mando de velocidades / Mando de velocidades lentas y rápidas*

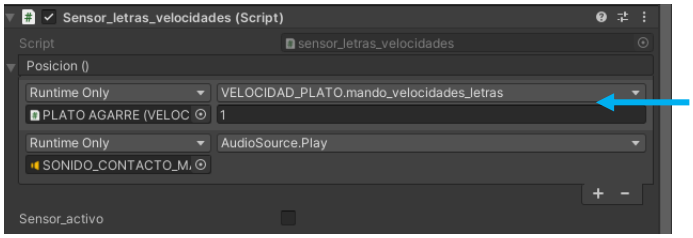
Para ello utilizaremos una de las funciones más demandas en nuestro proyecto, la función: *OnTriggerEnter*. Esta función nos va a permitir conocer cuando dos Box Collider colisionan entre sí. Para ello lo que haremos será crear un “Indicador”, que colisionará con cada una de las distintas posiciones del mando. Además de los Collider, uno de ellos debe contener un Rigidbody, y la opción *isTrigger* activada, de manera que cuando se produzca la colisión, no actúe la simulación física. En este caso, el componente *Rigidbody*

lo lleva asociado el indicador y la opción *isTrigger* está activada en el Collider de cada posición.

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.name == "INDICADOR_4")
    {
        sensor_activo = true;

        Posicion.Invoke();

        Debug.Log("sensor letra tocado");
    }
}
```

*Script 16. Función Void OnTriggerEnter*

Esta función está asociada dentro de un script, a cada una de las marcas o posiciones de los diferentes mandos que encontramos sobre la bancada (lo único que se modifica es el nombre del indicador que colisiona en cada caso). Cuando un objeto colisiona, la función recibe a ese objeto definido como *other*. De esta manera lo que hacemos es comprobar que other es el indicador del mando, en este caso el "INDICADOR_4". Si se verifica lo anterior, se activaría el evento *Posición* que utilizaremos para enviar la posición de fila o columna dentro de la matriz de velocidades que hemos declarado en el Husillo.




*Script 17. Evento de la función para la posición del mando de vel.*

En este caso, el mando de velocidades posiciona la columna dentro de la matriz de velocidades en la posición "1", ya que estamos leyendo la marca "B". El script que recibe dicha información es la función "mando_velocidades_letras ()" dentro del script "Velocidad_Plato" asociado al gameobject "Plato de Agarre".

```
public void mando_velocidades(int posicion)
{
    fila = posicion;
}

public void mando_velocidades_letras(int posicion_letra)
{
    columna = posicion_letra;
}
```

*Script 18. Posiciones de la matriz de velocidades*

### *5.4.1.2 Sistema de arranque y parada*

Cuando se produce la pulsación del botón de arranque, el script recibe la señal a través de la función "boton_arranque ()" que pone "funcionando" a "true" en caso de que se cumpla la condición que vemos en la figura.

```
public void boton_arranque(bool arranque)
{
    if (sensor_mampara2.sensor_activo == true && electricidad==true && funcionando==false)
    {
        funcionando = arranque;
        sonido_arranque.Invoke();
    }
```

*Script 19. Pulsador de arranque*

Una de esas condiciones hace referencia al sistema de seguridad de la máquina, que activa el freno del husillo en caso de levantar la mampara protectora mientras se realiza el mecanizado. Y la otra condición requiere que debe estar activada la corriente.

En el caso de la mampara hemos introducido dos Box Collider, uno sobre el asa de esta, y otro situado en la posición de mecanizado. Para la programación hemos optado de nuevo por introducir la función *OnTriggerEnter* y *OnTriggerExit* que detecta cuando la mampara entra y sale de nuestro sensor.

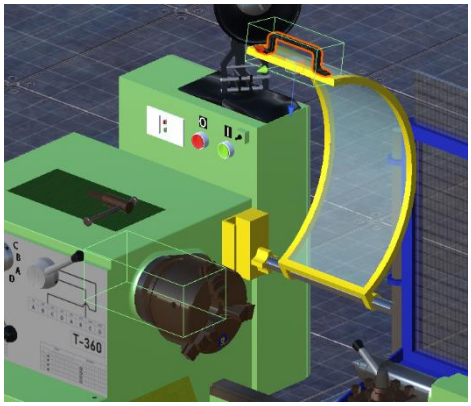


*Figura 88. Sensor mampara de protección*

Para leer el estado de la mampara desde el script que hemos creado en el Husillo, al tratarse de un gameObject distinto, lo que hacemos es asignar a una variable, la referencia de ese script. Esto se hace a través del comando *Get Component* , de manera que podamos leer y modificar en su caso, cuaquier variable pública dentro de ese gameObject.

Esto se hace declarando otra variable pública en la que asignaremos el gameObject al que hacemos referencia y por otro lado, una variable pública con tipo, el script al que hacemos referencia dentro de ese gameObject

```
public GameObject sensor_mampara;
private sensor_mampara_2 sensor_mampara2;

private void Awake()
{
    sensor_mampara2 = sensor_mampara.GetComponent<sensor_mampara_2>();

}
```

*Script 20. Función Void Awake*

Para la corriente, tanto el contacto general como la variable “electricidad” que recibe la pulsación del botón verde, deben estar a “true”. La programación de estos elementos sigue los mismos pasos anteriores.

Para programar la velocidad de rotación del Husillo utilizamos la función *void Update ()* donde leeremos la velocidad a partir la posición dentro de la matriz y realizaremos una

conversión a grados/segundo. El comando Time.deltaTime que vemos en el código permite ejecutar ese código n (unidades) /seg.

Para asignar una rotación a cualquier gameObject, accedemos a su transform y punto seguido hacemos referencia a la rotación de este. Esto se iguala a la rotación que queremos asignarle, en este caso debemos transformar la lectura de los grados en Euler a Cuaterniones, que es como trabaja la cinemática de Unity. La parada del Husillo se efectuará en el momento en el que alguna de esas variables cambie su posición a “false”.

```
void Update()
{
    if (electricidad_general.sensor_activo == true)
    {

        if (electricidad == true)
        {

            if (funcionando == true)
            {

                velocidad = velocidad_plato[fila, columna];
                x += (velocidad / 60) * 360 * Time.deltaTime;
                transform.rotation = Quaternion.Euler(x, 0, 0);


            }


        }

    }

}
```

*Script 21. Rotación del Husillo*

### *5.4.1.3 Mandril o Sistema de garras*

Para controlar la apertura de las garras mediante la llave del husillo hemos programado un conjunto de scripts que permiten introducir la llave en alguno de los orificios de este, y desplazar las garras según el sentido de giro de la llave.

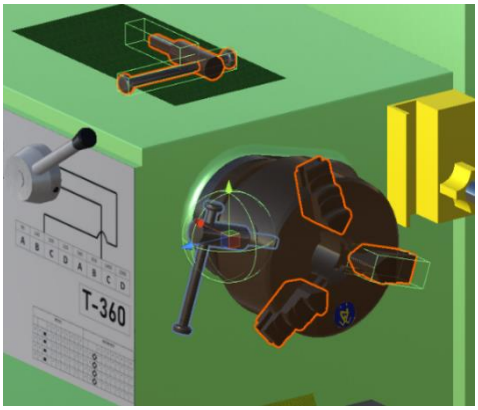
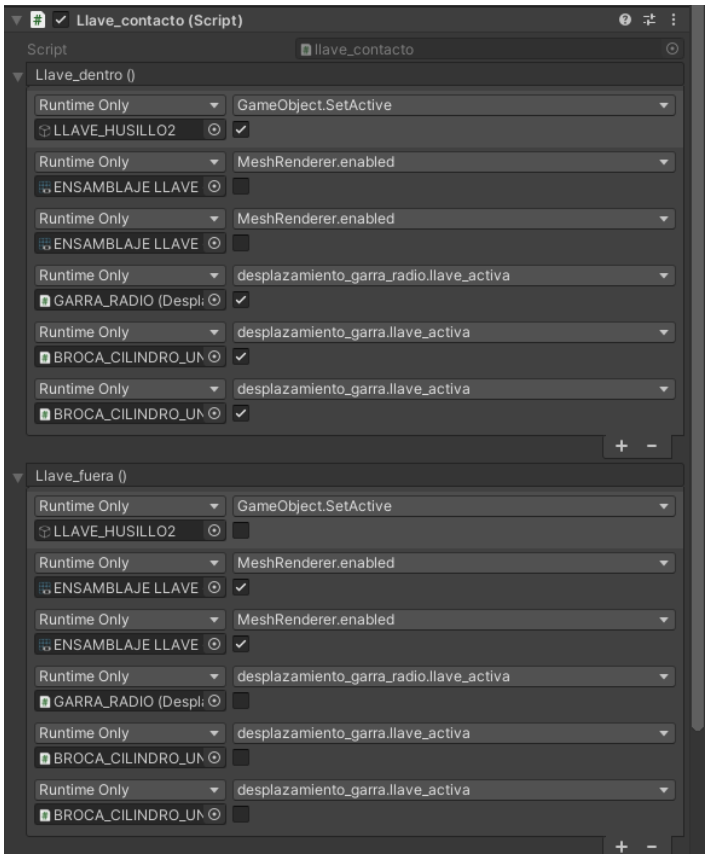



*Figura 89. Sistema de garras y llave de ajuste*

Alrededor del husillo hemos colocado un Capsule Collider (como el que aparece en la ilustración) por cada orificio. Así, podemos detectar la intención del usuario de introducir la llave en uno de ellos, a partir de la función *OnTriggerEnter*. Según si la llave entra o sale del Collider se producen los siguientes eventos:




*Script 22. Posición de la llave de ajuste*

En el evento de "Llave dentro" la llave colisiona con el Collider. Cuando esto tiene lugar, el script ejecuta las siguientes acciones:

- **Activa la llave fijada en el husillo** que corresponde a ese orificio a través del comando *Set Active* que permite activar o desactivar cualquier gameObject.
- **Esconde la llave principal** para que no haya duplicidad de llaves, desactivando el *Mesh Renderer*. En este caso no podemos utilizar *Set Active* para eliminar el gameObject ya que entonces el *Collider* no detectaría la pieza y por tanto que esta colisionando.
- **Permitir el desplazamiento de las garras.**

Por otro lado, una vez salga la llave de la posición del Collider, se activaría el evento "llave_fuera" que realizaría lo contrario de lo que acabamos de mencionar

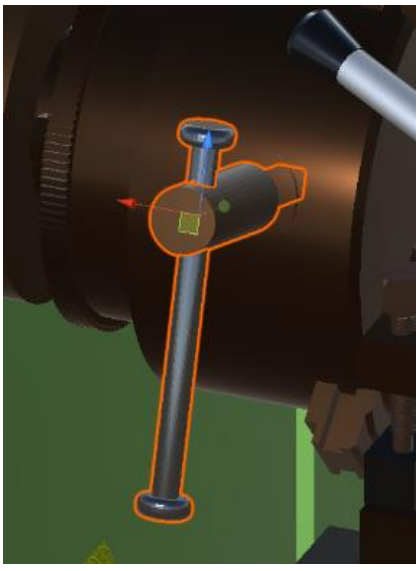
A través del siguiente script leemos la rotación de la mano que estamos utilizando a la hora de colocar la llave dentro de uno de los tres orificios que tiene el Husillo y le decimos a la llave que gire siguiendo dicha rotación. Para ello utilizamos el comando *Quaternion.AngleAxis* mediante el cual asignamos a la llave la rotación de nuestra mano sobre el eje "y" de la llave. Además, como la llave fijada al Husillo presenta una inclinación inicial, debemos sumarla a la rotación propia de la mano. La variable "estado" determina que mano está cogiendo la llave.

```
void Update()
{


    if (estado.ultimo_estado == 0)
    {


        if (estado.estado == 1)
        {
            transform.rotation = Quaternion.AngleAxis(mano_mov_drch.rotation.eulerAngles.z, direccion_y) * rotacionInicial;
        }

        else if (estado.estado == 2)
        {
            transform.rotation = Quaternion.AngleAxis(mano_mov_izq.rotation.eulerAngles.z, direccion_y) * rotacionInicial;
        }
    }
    else
    {

     if (estado.ultimo_estado == 1)
        {
            transform.rotation = Quaternion.AngleAxis(mano_mov_drch.rotation.eulerAngles.z, direccion_y) * rotacionInicial;
        }

        else if (estado.ultimo_estado == 2)
        {
            transform.rotation = Quaternion.AngleAxis(mano_mov_izq.rotation.eulerAngles.z, direccion_y) * rotacionInicial;
        }
    }



}
```

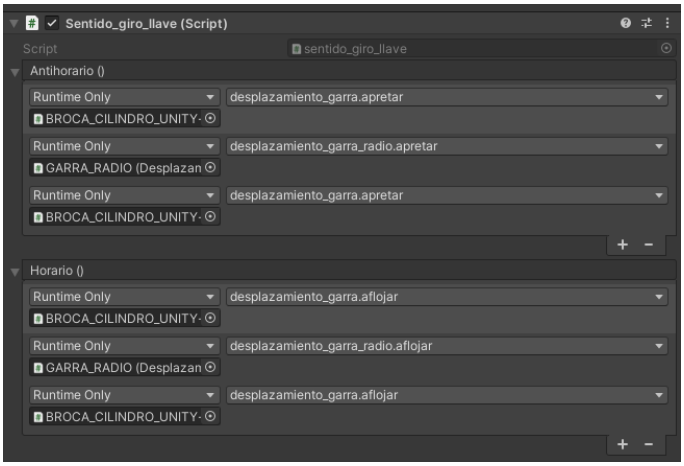
*Script 23. Giro de la llave*

Ahora debemos traducir la rotación de la llave en el desplazamiento de las garras. Para ello realizaremos otro script que se basará en calcular el producto vectorial de dos vectores. Estos vectores en realidad se refieren al eje “z” (eje azul) de la llave, que varía su dirección conforme se produce el giro de la llave. Dependiendo del sentido del vector normal, obtendremos el sentido de giro.

```
void Update()
{
    vector_normal = Vector3.Cross(posicion_inicial, transform.forward);

    if (vector_normal.y > 0f)
    {
        cont1++;
        if (cont1 == 5)
        {
            Debug.Log("sentido antihorario");
            antihorario.Invoke();
            cont2 = 0;
            cont1--;
        }

    }
    else if (vector_normal.y < 0f)
    {
        cont2++;
        if (cont2 == 5)
        {
            Debug.Log("sentido horario");
            horario.Invoke();
            cont1 = 0;
            cont2--;
        }
    }
    else
    {
        Debug.Log("no hay rotacion");
    }

    posicion_inicial = transform.forward;

}
```



*Script 24. Sentido de Giro*




*Script 25. Eventos del sentido de giro de la llave*

Por otro lado, los eventos que hemos introducido se comunicarían con las garras, indicando a estas el sentido de giro en el que deben desplazarse. El desplazamiento se produce siempre y cuando exista alguna llave dentro de uno de los orificios y las garras no hayan agarrado la pieza.

```
public void apretar()
{
    if (activada == true && tope.tope==false)
    {
        transform.Translate(0, 0, 0.01f * Time.deltaTime);
    }
}

public void aflojar()
{
    if (activada == true)
    {
        transform.Translate(0, 0, -0.01f * Time.deltaTime);
    }
}
```

*Script 26. Desplazamiento de las garras*

En cuanto a la pieza, esta se introduce en el husillo a través de un Collider que detecta cuando el usuario pretende introducir la pieza, de manera que cuando la suelta, esta queda fijada en una posición centrada en el husillo.

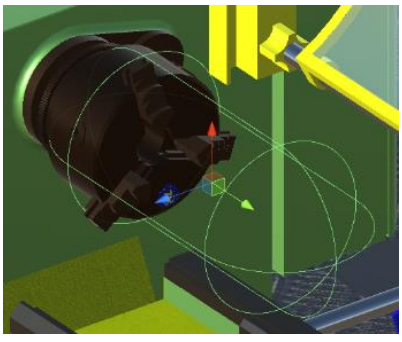


*Figura 90. Sensor de posición de la pieza en el husillo*

Para que la experiencia sea lo más realista posible, programaremos las garras para que estas tengan que estar lo suficientemente apretadas, de manera que la pieza quede sujeta y no caiga sobre la bancada en caso de que la apertura sea muy holgada. Para ello podemos modificar las opciones del componente Rigidbody de la pieza que vamos a mecanizar.

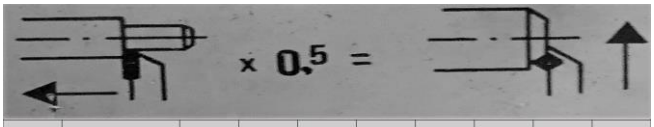
```
if(posicionactual.y < radio_pieza.radio * 100 + 0.3f && posicionactual.y > radio_pieza.radio * 100)
{
    sensor_posicion_pieza.SetActive(true);

    if(sensor.entra == true)
    {
        cinematico.isKinematic = true;

    }

}

else if(posicionactual.y > radio_pieza.radio * 100 + 0.3f)
{
    sensor_posicion_pieza.SetActive(true);

    if (sensor.entra == true && agarrando == false)
    {
        cinematico.isKinematic = false;
        gravedad.useGravity = true;

    }
}

else if(posicionactual.y < radio_pieza.radio * 100 && sensor.entra == false)
{
    sensor_posicion_pieza.SetActive(false);

}
```

*Script 27. Modificación de las componentes cinemáticas*

### 5.4.2 Carro longitudinal

Para el desplazamiento del carro longitudinal, reutilizaremos la función anterior que permite leer el sentido de giro del volante del carro, de manera que este se desplace en un sentido o en otro. En este caso, para que el desplazamiento sea más fluido, calcularemos el módulo del vector normal resultante de manera que el desplazamiento del carro se produzca para giro mínimo del volante.

En el caso de utilizar la palanca de cilindrar y refrentar para el desplazamiento automático del carro, hemos colocado dos Collider que detectan cuando se quiere cilindrar o refrentar en función de con cuál de ellos colisione la palanca. Para ello se vuelve a recurrir a la función *OnTriggerEnter*. Para la selección de avances creamos otra matriz de igual manera que hemos hecho con las velocidades del Husillo.




| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | X | ● | 0,045 | 0,051 | 0,054 | 0,056 | 0,062 | 0,068 | 0,073 | 0,079 |
| 2 | X | ● | 0,09 | 0,102 | 0,108 | 0,113 | 0,125 | 0,136 | 0,147 | 0,159 |
| 3 | X | ● | 0,181 | 0,204 | 0,216 | 0,227 | 0,250 | 0,272 | 0,295 | 0,318 |
| 4 | X | ● | 0,363 | 0,409 | 0,432 | 0,454 | 0,5 | 0,545 | 0,591 | 0,636 |

*Figura 92. Tabla de avances*

Si la palanca colisiona con el sensor de cilindrar, se activaría la función que activa el desplazamiento del carro en función del avance que hayamos seleccionado en la bancada. Este se moverá dentro de un intervalo que irá fijado por los topes de la bancada configurados también como Colliders.

```
void Update()
{
    if(  direccion_avanc==1 && tipo_rosca==1 && plato_arrancado.funcionando==true)
    {
        rpm = revoluciones.velocidad;
        avance = avance_cuchilla[fila, columna];
        desplazamiento_carro = (avance * rpm) / 60;

        if (sensor_cilindrado.sensor_activo == true && tope1.tope==false)
        {
            transform.Translate(desplazamiento_carro / 1000, 0, 0);
            cilindrado_activado.Invoke();
        }


    }


}
```

*Script 28. Desplazamiento del carro longitudinal*

Los avances de la tabla están expresados en mm/rev de manera que debemos hacer las conversiones necesarias para expresar el avance en m/s. Para ello debemos leer la velocidad de rotación que se ha seleccionado para el husillo haciendo referencia al script de este mediante el comando *Get Component*.

### 5.4.3 Carro transversal

Este tiene el mismo principio de funcionamiento que el caso anterior. Los avances se seleccionan del mismo modo, salvo que en el refrentado estos son la mitad. También se puede variar el sentido de avance del carro transversal a través de la palanca de sentido de avance situada en la bancada. En función de la posición de la palanca, el carro avanzará hacia delante o hacia atrás.

```
void Update()
{
    if(arrancado.funcionando==true && sensor.sensor_activo==true)
    {
        if(sentido_avance==0 && tope.sensor_activo2==false && cilindrado==false)
        {
            avance_profundidad = velocidad_avance.desplazamiento_carro * 0.5f;

            transform.Translate(0, 0, -avance_profundidad / 1000);
        }

        else if(sentido_avance==1 && tope.sensor_activo1==false || sentido_avance==0 && tope.sensor_activo1==false && cilindrado==true)
        {
            avance_profundidad = velocidad_avance.desplazamiento_carro * 0.5f;

            transform.Translate(0, 0, avance_profundidad / 1000);
        }
    }

}
```

*Script 29. desplazamiento del carro transversal*

El torno tiene un sistema de seguridad mediante el cual se impide que la cuchilla avance en sentido transversal, en caso de que se esté cilindrando. Si por error, el usuario mueve la palanca de cilindrar y refrentar hacia arriba, y el sentido de avance transversal es hacia delante, el carro se movería hacia detrás para así evitar un accidente.

### 5.4.4 Carro portaherramientas

Este solo se moverá de manera manual a través del volante del carro portaherramientas. De esta manera volvemos a hacer uso de la función anterior que nos permite leer el sentido de giro del volante y nuestro carro podrá moverse en un sentido u otro limitado por los topes que hemos introducido.

```
public void avance_positivo()
{


    if (mano_drch.activeSelf == true && tope.sensor_activo2==false || mano_izq.activeSelf == true && tope.sensor_activo2 == false)
    {

        transform.Translate(Time.deltaTime * 0.04f, 0,0 );
    }
}

public void avance_negativo()
{


    if (mano_drch.activeSelf == true && tope.sensor_activo1==false || mano_izq.activeSelf == true && tope.sensor_activo1 == false)
    {

        transform.Translate(-Time.deltaTime * 0.04f, 0,0 );
    }
}
```

*Script 30. Desplazamiento carro portaherramientas*

### 5.4.5 Carro del contrapunto

El contrapunto presenta un box Collider que permite interactuar con él para desplazarlo a través de las guías de la bancada. Cuando la mano agarra el Collider, este se desplaza siguiente el desplazamiento de nuestra mano.

Para el bloqueo del contrapunto, se utiliza la palanca del contrapunto que cuando colisiona con el sensor que hemos introducido en el recorrido final de la palanca, se activa un evento que desactiva el Collider del contrapunto. De esta manera el usuario no podría interactuar con el carro.

```
void Update()
{
    if(sensor_bloqueo.bloqueo==true )
    {
        gameObject.GetComponent<BoxCollider>().enabled = false;
    }

    else if(sensor_bloqueo.bloqueo==false)
    {
        gameObject.GetComponent<BoxCollider>().enabled = true;
    }
}
```

*Script 31. Bloqueo del carro del contrapunto*

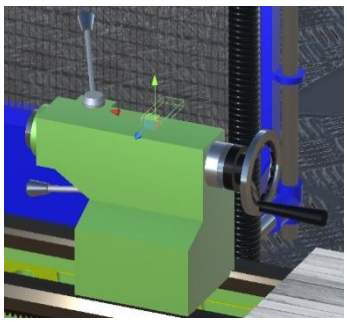


*Figura 93. Sensor de la palanca de bloqueo*

Asimismo, se utiliza la función de sentido de giro, que permite desplazar el eje del contrapunto. Este eje puede desplazar la broca de centrar o la punta del contrapunto de manera que podemos intercambiar estos objetos según el usuario lo desee. Para ello hemos introducido un Collider en el eje, que actúa como sensor y detecta la entrada de

estos gameObjects. Si el usuario se acerca con la broca de centrar a este Collider y la suelta, esta queda fija sobre el eje. Lo mismo para la punta del contrapunto.

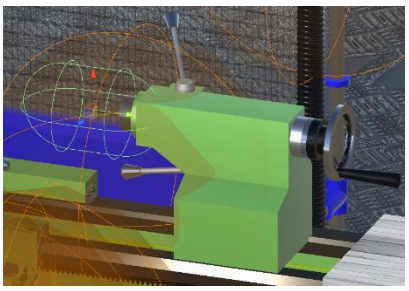


*Figura 94. Sensor de posición de la caña del contrapunto*

```
if (other.gameObject.name == "PIEZA1_EJE_TALADRO_UNITY-1 (PIEZA1_EJE_TALADRO_UNITY)")
{
    if (cambio_caña.sensor_activo1 == true)
    {
        fijado = 2;
        dentro = true;
    }
}

else if (other.gameObject.name == "BROCA_CONTRAPUNTO")
{
    if (cambio_caña.sensor_activo1 == true)
    {
        fijado = 1;
        dentro = true;
    }

}


vate void OnTriggerExit(Collider other)

if (other.gameObject.name == "PIEZA1_EJE_TALADRO_UNITY-1 (PIEZA1_EJE_TALADRO_UNITY)" || other.gameObject.name == "BROCA_CONTRAPUNTO")
{
    dentro = false;
    fijado = 0;
}
```

*Script 32. Detección de la caña del contrapunto/ broca de centrar*

Para que ahora la caña de la broca de centrar se desplace de manera solidaria con el eje, debemos utilizar el comando *Set Parent*. Este comando, como su nombre indica, permite asignar un padre al gameObject que introduzcamos dentro del eje. En este caso se le asigna el eje del contrapunto de manera que se desplacen conjuntamente al girar el volante.

```
public void fijar()
{
    if (sensor.fijado == 2 && sensor_caña.bloqueo==false)
    {
        cinematico.isKinematic = true;
        transform.position = new Vector3(-1.54139996f+desplazamiento_relativo, 1.16470003f, -1.35423994f);
        transform.rotation = Quaternion.Euler(0, 0, -90);
        transform.SetParent(eje_contrapunto);
    }
}
```

*Script 33. Función para fijar la caña en el carro del contrapunto*

Para finalizar con el contrapunto, la manivela situada sobre este nos permite fijar la caña al eje. Para ello se vuelve a utilizar el comando anterior para dejar "huérfano" al gameObject que se esté utilizando (broca o contrapunto) sobre el Prefab que permite interactuar con él y de esta manera el sistema de agarre no lo detectaría.

### 5.4.6 Pieza de mecanizado

Los ejes que se van cilindrar serán generados a partir de la entrada de usuario, a través de un panel en el que este pueda especificar las dimensiones de la pieza. Estos datos serán recogidos dentro de un script que guardará los datos de altura y radio para generar la pieza. Hemos establecido un límite máximo de 25cm de altrua y 5cm de radio.

```
public void sumar_altura()
{
    if (altura < 0.25f)
    {
        altura=altura+0.01f;
    }
}
```

```
public void restar_altura()
{
    if(altura>0.05f)
    {
        altura=altura-0.01f;
    }
}
```


```
public void restar_radio()
{
    if(radio>0.02f)
    {
        radio=radio-0.01f;
    }
}
```

```
public void sumar_radio()
{
    if(radio<0.05f)
    {
        radio=radio+0.01f;
    }
}
```

*Script 34. Entrada datos altura y radio*

Para generar la pieza hemos optado por construir el eje a partir de pequeñas chapas cilíndricas de 1mm de espesor que incorporan un Mesh Collider dentro de cada Prefab. De esta manera, si queremos modificar su radio, bastará con modificar la escala de cada una de las chapas. Esto se hará accediendo a la escala dentro del transfrom de cada Prefab, mediante el comando *transfrom.LocalScale*. en este caso debemos modificar el eje "x" y el eje "z".

```
public void formar_pieza()
{
for (float a = 0; a < altura; a = a + 2 * cilindro_base.transform.localScale.y)
{
GameObject pieza = Instantiate (cilindro_base, new Vector3(cubo_referencia.transform.position.x,
cubo_referencia.transform.position.x + a, cubo_referencia.transform.position.z),
Quaternion.identity);
pieza.transform.localScale = new Vector3(2 * radio, cilindro_base.transform.localScale.y, 2 *
radio);
pieza.transform.parent = cilindro_padre.transform;
}

}
```

*Script 35. Función para crear la pieza*

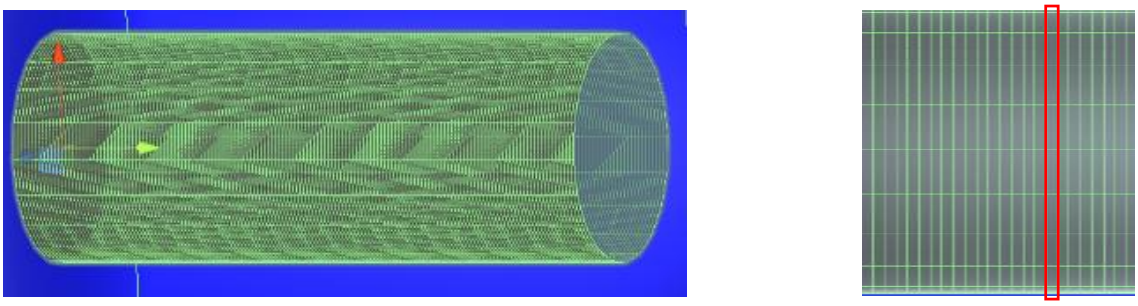



*Figura 95. Pieza Generada*

En cuanto a la altura, bastará que introducir tantas chapas como altura tenga nuestro cilindro. Para ello hemos creado un bucle *for* en el que se utiliza el comando *Instantiate* que nos permite crear objetos en la escena a través de un plantilla o Prefab. Esto nos generará una pieza situada dentro un gameObject con el que poder interactuar para coger nuestra pieza y situarla dentro del plato

### 5.4.7 Cuchilla

Para simular el mecanizado de la pieza realizaremos un script que modifique la escala de cada una de las chapas que entren en contacto con la cuchilla. La cuchilla tiene un Collider en el filo de manera que conforme detecta los Prefab que componen el cilindro, se ejecutaría la función *OnTriggerEnter*. Esta función recibe el gameObject que colisiona y lo guarda como "other", como ya hemos explicado anteriormente. De esta manera solo debemos modificar la escala de cada uno de los gameObject que penetran en función del avance de nuestra cuchilla dentro de la pieza.

```
other.gameObject.transform.localScale = new Vector3(other.gameObject.transform.localScale.x *
factor_escala, other.gameObject.transform.localScale.y, other.gameObject.transform.localScale.z
* factor_escala);
```

*Script 36. Mecanizado del material de la pieza*

Para ello hemos creado un factor de escala que calcula el porcentaje que debe reducirse el diametro actual de la pieza en función del avance de la cuchilla.

```
factor_escala = (other.gameObject.transform.localScale.x - 2*avance_herramienta)
/ (other.gameObject.transform.localScale.x);
```

*Script 37. Factor de escala*

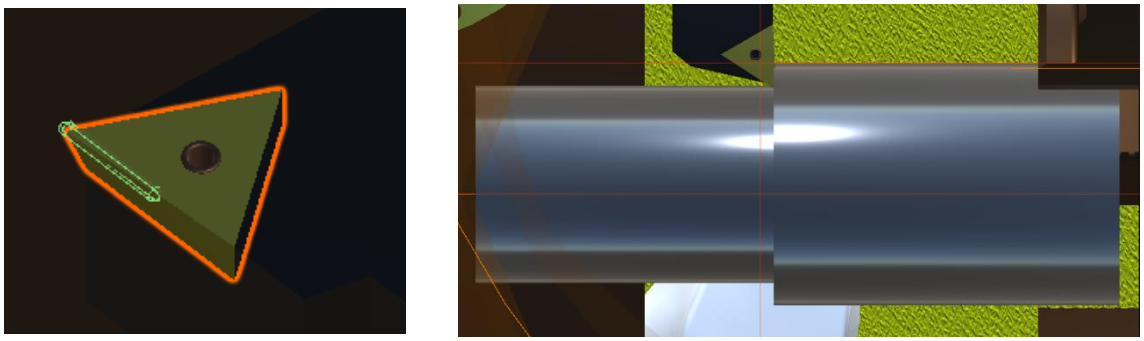


*Figura 96. Plaquita de la cuchilla*

### 5.4.8 UI, materiales y sonido

#### *5.4.8.1 User Interface (UI)*

Para mejorarla experiencia del usuario dentro de la simulación utilizaremos esta herramienta que sirve de comunicación con el usuario a través de imágenes o textos flotantes en nuestra escena.

- **Cuadro digital**

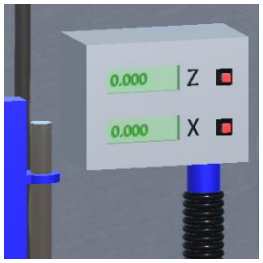



*Figura 97. Cuadro digital*

Con la nueva normativa reguladora de este tipo de tornos se inocopora un display sobre la bancada que mide el desplazamiento de la cuchilla en sentido longitudinal y transversal, de manera que sirve de gran ayuda al profesional que utiliza la máquina para conocer las medidas de avane en profundidad y longitud de mecanizado.

En este caso, utilizaremos la herramienta UI para introducir un texto que muestre por pantalla la lectura del desplazamiento de la cuchilla en todo momento. Para ello debemos irnos a la pestaña superior **GameObject > UI > TextMeshPro**.

Esto nos creará automáticamente el *Canvas* dentro de la escena. Este componente se encargará de renderizar los objetos de interfaz de usuario, por lo que cada uno de ellos deberá estar dentro de este para que puedan visualizarse.

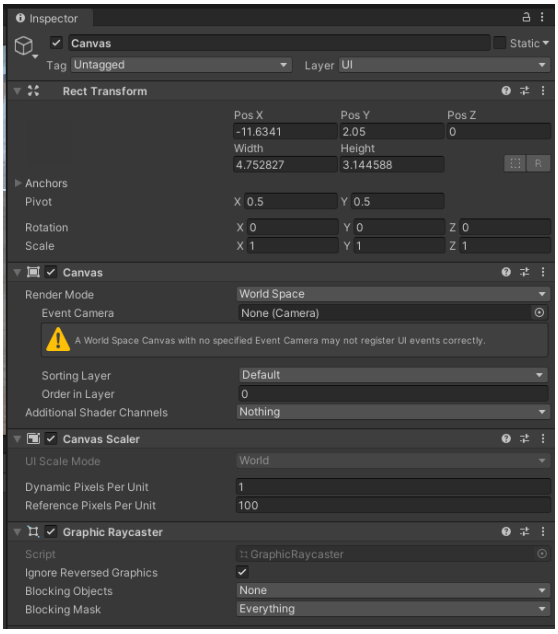



*Figura 98. Canvas*

Seleccionando el objeto podemos modificar la posición y la escala de este a través de la pestaña **Rect Transform**. Dentro de la pestaña **Canvas** debemos modificar el modo de renderizado a *World Space* de manera que la imagen se procese como un objeto plano, pero no sea necesario enfrentar a la cámara al plano de la imagen, sino que podamos observarla desde otros ángulos. Por otro lado, el **Canvas Scaler** se encarga del escalado y la densidad total de pixeles de los elementos. Finalmente, el **Graphic Raycaster** controla cuando algún elemento gráfico dentro del Canvas ha sido golpeado. Esto último sirve de utilidad si se quiere interactuar con un panel que introduce botones o barras. En este caso nosotros introduciremos botones "físicos" de manera que no se utilizará.

Como hijos de este gameObject aparecerá nuestro texto. A través del Inspector, de nuevo, podemos modificar la posición dentro del Canvas, así como editar la configuración del propio texto, aplicando texturas, colores, tipografía, etc.

Para poder acceder a nuestro TextMeshPro desde un script debemos declarar una nueva librería: `using TMPro`. Que incorpora las utilidades necesarias para comunicarnos con nuestro texto.

```
public TMP_Text pantalla_ejex;

void Update()
    {
        posicion_relativa = transform.position - posicionInicial;
        posicion_relativa_mm = posicion_relativa * 1000;
        pantalla_ejex.text = posicion_relativa_mm.x.ToString("F3");
        lastPosition = transform.position;

    }
```

*Script 38. Variable de texto*

Para poder escribir un texto a través de un script debemos declarar una variable de tipo *"TMP_Text"* como se muestra en el script que hemos asociado a la cuchilla para poder leer el desplazamiento relativo de esta. Ahora podemos acceder al texto con *"pantalla_ejex.text"* y mediante el comando *ToString* se envía la variable que queremos leer por pantalla. Entre paréntesis "F3" se indican los decimales que introduce nuestro texto.De la misma forma procederíamos con el desplazamiento en el eje "z" o transversal.

El cuadro digital incorpora dos pulsadores a partir de los cuales el usuario puede posicionar la pieza cuando la cuchilla toca con esta. Esto hace que la última posición que se haya guardado se asigne a la posición inicial.

- **Plato de Agarre**

Mediante la herramienta de texto podemos visualizar en la escena el radio de apertura del sistema de agarre de manera que podamos comprobar cuando la pieza está bien sujeta. Para ello, ahora leemos el desplazamiento de una de las garras cuyo eje vertical está alineado con el propio eje vertical del plato. De esta manera declaramos una variable que almacene la posición relativa de la garra y para su salida por pantalla realizamos los mismos pasos anteriores.

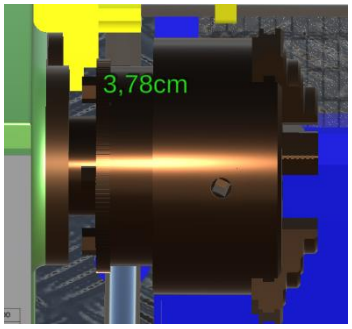



*Figura 99. Radio de apertura del Husillo*

- **Tabla de avances**

A parte de textos, la herramienta de UI también permite renderizar imágenes seleccionando **UI > Raw Image**. Ahora dentro del Canvas visualizaríamos también la imagen que hayamos introducido. Para introducirla, dentro del Inspector encontramos la pestaña **Raw Image,** en la opción *Texture* bastaría con arrastrar la imagen. El software permite introducir varios formatos, en nuestro caso hemos importado el archivo en formato JPEG.

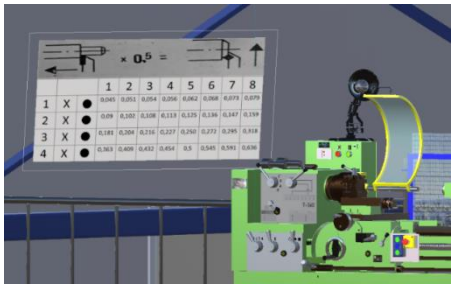


*Figura 100. Tabla de avances*

*5.4.8.2 Materiales*

Otro aspecto importante es dotar de un aspecto realista a los elementos de nuestra máquina, así como al entorno en donde se encuentra. Para ello Unity nos brinda la posibilidad de crear materiales que se pueden aplicar de forma sencilla sobre cualquier gameObject o Prefab.

Para crear un material debemos irnos a la ventana de Project, y pulsando botón derecho **Create > Material**. De esta forma se nos abre un nuevo material que podemos modificar a nuestra conveniencia a partir de las configuraciones dentro de la ventana de Inspector.

En primer lugar, encontramos el **Shader**, que determina las propiedades que muestra el inspector de un material. Este es un tipo especializado de programa gráfico que determina cómo se combina la información de textura e iluminación para generar los píxeles de GameObject representado en pantalla.


*Figura 101. New Material*

En este caso hemos seleccionado el Shader Standard con el que podemos recrear casi cualquier material del mundo real.

El primer parámetro que podemos modificar dentro de este shader, es el **Rendering Mode** con el que podemos determinar si nuestro objeto es traslúcido y como se quiere visualizar en escena.

- **Opaque**. Es el valor predeterminado y es adecuado para objetos sólidos normales sin áreas transparentes.
- **Cutout**. Permite crear un efecto transparente que tiene bordes duros entre las áreas opacas y transparentes. En este modo, no hay áreas semitransparentes, la textura es 100% opaca o invisible. Esto es útil cuando se utiliza la transparencia para crear la forma de materiales como hojas, o tela.
- **Transparent**. Adecuado para renderizar materiales transparentes realistas como plástico transparente o vidrio. En este modo, el material en sí asumirá valores de transparencia (basados en el canal alfa de la textura y el alfa del color de tinte), sin embargo, los reflejos de iluminación permanecerán visibles a plena claridad como es el caso de los materiales transparentes reales.
- **Fade**. Permite que los valores de transparencia desvanezcan por completo un objeto, incluidos los resaltados o reflejos especulares que pueda tener. Este modo es útil si desea animar un objeto que se desvanece dentro o fuera. No es adecuado para renderizar materiales transparentes realistas como plástico transparente o vidrio, ya que los reflejos y reflejos también se desvanecerán.

Luego tenemos el **Albedo**. Este parámetro controla el color base de la superficie del objeto en donde se aplique este material. Se puede hacer a través de la pestaña **Color** donde podemos modificar el color del material en una escala RGB o a través de una

textura que se aplique sobre el albedo. Esto último resultará muy útil a la hora de visualizar el esquema de los mandos de la bancada. Para ello hemos creado una textura previamente que se asemeje al panel real, y seguidamente la hemos aplicado a nuestro albedo siendo este el resultado (Figura).

Otro de los aspectos que nos resultará útil modificar es el parámetro Metálico (**Metalic**). Cuando se trabaja en el flujo de trabajo Metálico (a diferencia del flujo de trabajo especular), la reflectividad y la respuesta de luz de la superficie se modifican por el nivel metálico y el nivel de suavizado o **Smoothness**. Cuando una superficie es más metálica, refleja más el entorno y su color albedo se vuelve menos visible.

Por último, otro de los aspectos que utilizaremos para configurar nuestros materiales será el parámetro **Normal Map**. Esto se utiliza cuando se quiere aplicar textura sobre la superficie del material, de manera que podemos introducir golpes, ranuras, arañazos, o cualquier tipo de forma que captan la luz como si estuvieran representados por geometría real.

Imaginemos un elemento de la superficie lateral de un cilindro representado por tres planos, visto desde la planta. Las flechas naranjas que atraviesan los polígonos alrededor del exterior del cilindro representan las normales de superficie. Estos son los valores utilizados para calcular cómo se refleja la luz fuera de la superficie, de modo que el sombreado da la impresión de una curva suave (como se representa por la curva verde). Esto no afecta a la naturaleza poligonal real de la malla, solo a cómo se calcula la iluminación en las superficies planas. Esta superficie curva aparente no está realmente presente, y la visualización de las caras en ángulos de mirada revelará la verdadera naturaleza de los polígonos planos, sin embargo, desde la mayoría de los ángulos de visión el cilindro parece tener una superficie curva lisa.

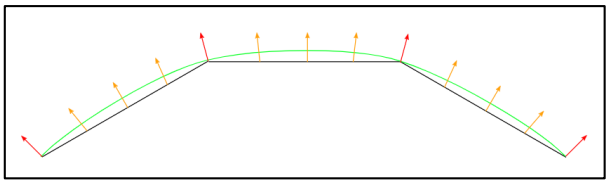


*Figura 102 Normales de superficie*
*Fuente: Unity - Manual: Normal map (Bump mapping) (unity3d.com)*

Para cambiar la apariencia de este cilindro, se introduce un normal map que modifica las normales de superficie a través de la superficie del modelo. Esto es, una textura de imagen asignada a la superficie de un modelo, similar a las texturas de color normales, sin embargo, cada Pixel en la textura del mapa normal (llamado elemento de textura) representa una desviación en la dirección normal de la superficie lejos de la superficie "verdadera" normal del polígono plano (o interpolado suave). En este diagrama, cada flecha naranja corresponde a un píxel en la textura normal Map.

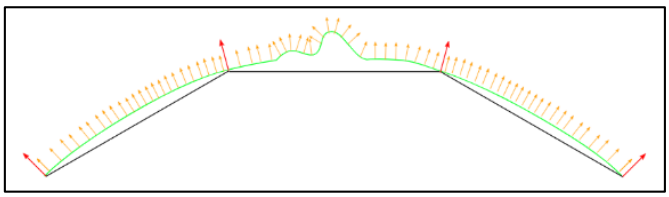


*Figura 103. Mapa normal. Fuente: Unity - Manual: Normal map (Bump mapping)*

Para conseguir esto, los programas que permiten transformar una textura cruda en un archivo normal Map, lo que hacen es representar los valores de RGB de cada elemento de textura en los valores X, Y y Z de un vector de dirección que se aplica a las modificaciones de las normales de superficie.

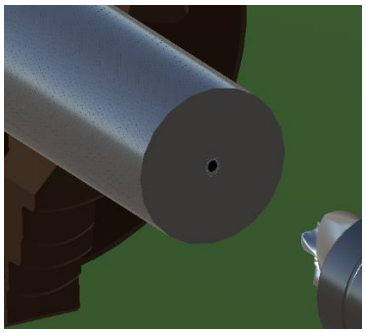


*Figura 104. Ejemplo de mapa normal sobre la pieza*

La broca de centrar presenta un script mediante el cual se cambia el mapa de texturas de la pieza que estamos mecanizando, de manera que simula realizar la operación de taladrado.

### *5.4.8.3 Sonido Ambiente*

Para seguir mejorando la experiencia virtual, se han introducido archivos de sonido los cuales fueron tomados directamente desde el laboratorio, y que permiten introducirnos, de manera más realista, dentro de la simulación que hemos creado.

Para crear un archivo de sonido debemos ir a **GameObject > Audio > AudioSource**. De esta manera se introduce en GameObject en el que podemos introducir clips de audio para ser reproducidos durante el proyecto.

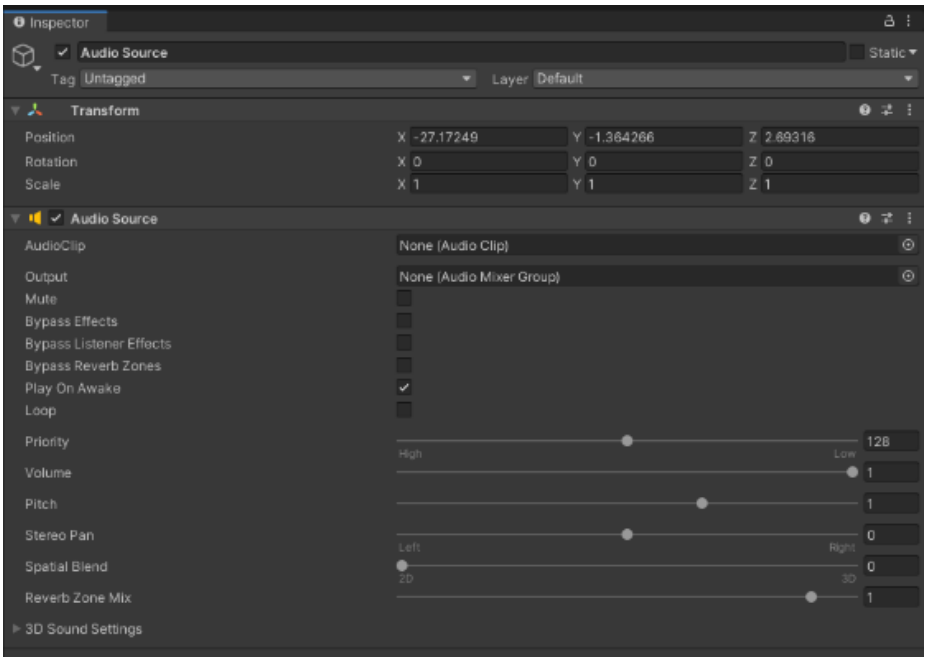
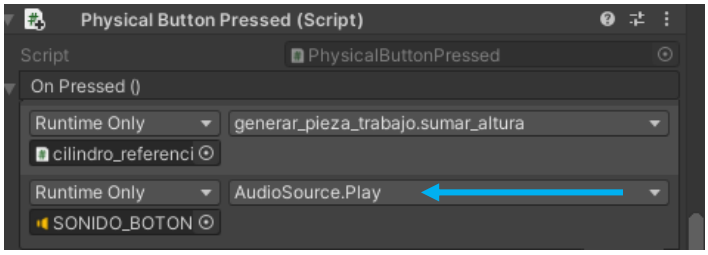



*Figura 105. AudioSource*

Este es el aspecto que tiene el configurador de sonido dentro de la pestaña Inspector. En la opción *Audio Clip* podemos introducir nuestro archivo de audio (en nuestro caso en formato MP3). Entre las opciones más destacadas, usaremos la opción *Loop* mediante la cual podemos repetir un mismo Clip de audio en bucle. También está la opción *Play On Awake* que ejecuta la reproducción del audio nada más comenzar la aplicación.

A la hora de acceder al archivo de sonido dentro de un script podemos hacer una referencia con el comando *Get Component* declarando una variable de tipo *AudiSource* o, por otra parte, también resulta útil crear un evento y asignar la función que queremos realizar.

Atendiendo a la segunda opción, imaginemos que queremos interactuar con un archivo de sonido cuando pulsamos un cierto botón. Para ello, en el evento *OnPressed* de nuestro botón introducimos el gameObject del archivo de sonido.




*Script 39. Physical Button Pressed*

Mediante esa función activaríamos la función Play del *AudioSource*, pero se pueden realizar otras muchas como se muestran en la pestaña desplegable. Aparte de esta, dentro de la aplicación utilizaremos la función *Pause ()* y *PlayOnDelayed ()*, que nos permitirán pausar el archivo de sonido o reproducirlo con un cierto retraso.

# CAPÍTULO 6. CONCLUSIONES

La Realidad Virtual lleva varios años queriendo asentarse con fuerza como una herramienta tecnológica que permita mejorar el desarrollo técnico-económico de la humanidad. En ese sentido, esta tecnología tiene una proyección de crecimiento bastante alta y cada vez son más las empresas las que buscan introducir la realidad virtual dentro de sus líneas estratégicas.

Este proyecto ha servido para dar cuenta de las diferentes aplicaciones que se pueden llevar a cabo a través de la Realidad Virtual, en este caso mediante la simulación de un torno con fines educativos.

## 6.1 Objetivos cumplidos

El resultado que se ha obtenido a partir de la realización del proyecto ha sido favorable. En materia de modelado, a través del software en 3D de SolidWorks, se ha conseguido crear un torno con un aspecto y presencia bastante similares a su modelo real, de forma que el usuario que lo maneja es capaz de identificarlo con los tornos presentes en el ELDI.

En cuanto a su programación, la realidad virtual no entiende de limitaciones, en este caso Unity nos ha permitido programar con éxito la mayoría de funciones que podemos llevar a cabo en el torno del laboratorio de fabricación. En concreto, las operaciones que se pueden llevar a cabo dentro del torno virtual serán cilindrados exteriores, así como refrentados. También se consigue simular la operación de taladrado en la pieza a través de la broca de centrar.

Finalmente, el entorno de inmersión permite al usuario evadirse del mundo real para introducirse de lleno en la simulación, engañando a nuestros sentidos con la utilización de una nave industrial que se ha ambientado a partir de elementos de sonido y otros materiales.

Todo ello ha permitido crear una aplicación en soporte digital para que el alumno de la Universidad Politécnica de Cartagena pueda conocer el funcionamiento básico de este tipo de tornos a través de un entorno virtual.

## 6.2 Limitaciones

Como se ha dicho anteriormente, la realidad virtual no entiende de limitaciones, sin embargo, al trabajar con equipos de gama media, el potencial que podemos extraer de esta tecnología puede verse limitado. Es por ello que la realidad virtual solo pueda utilizarse en aquellos equipos que cuenten con las especificaciones que hemos comentado a lo largo del desarrollo del trabajo. Esto convierte a la realidad virtual en una tecnología todavía poco accesible para el público convencional debido al alto coste que supone adquirir un ordenador de estas características además del propio equipo de realidad virtual.

Otro de los problemas que se han observado con la puesta en marcha de la aplicación tiene que ver con la adaptación de aquellos usuarios que no están acostumbrados a trabajar con realidad virtual. Puede producir defectos de equilibrio en el organismo o incluso ansiedad debido a las exigencias que esperamos de la situación de experimentación.


*Figura 106. Imagen tomada en Unity de la aplicación*

# CAPÍTULO 7. BIBLIOGRAFÍA

[1] REAL ACADEMIA ESPAÑOLA: *Diccionario de la lengua española*, 23.ªed., [versión 23.3 en línea]. < https://dle.rae.es/realidad?m=form#CfxhrOR > [consulta: 2 de noviembre 2020].

[2] R.W. Sherman, B.A. Craig. *Understanding Virtual Reality*. USA, San Francisco: Elsevier Science, 2003.

[3] J. Boo Gustems. *Introducción a la realidad virtual*. Capítulo 3. Universitat Politécnica de Catalunya. [En línea]. Disponible en JaviBoo.pdf (upc.edu) [consulta: 3 de noviembre 2020].

[4] M. Santamaría. ¿Qué es la realidad virtual? XperienciaVirtual. Disponible en: https://www.xperienciavirtual.es/es/que-es-la-realidad-virtual/ . [consulta: 8 noviembre 2020]

[5] E. Cárdenas, L. Morales, A. Ussa. "Las estereoscopía, métodos y aplicaciones en diferentes áreas del conocimiento". Revista Científica *"General José María Córdova"*. Bogotá, Colombia: 2015, Vol. 13, núm. 16, pp. 201-219.

[6] A. Gifreu. "La realidad virtual. Como afectará a los webdocs". Parte 2 Webdocs. Historias del siglo XXI. RTVE. Disponible en: https://blog.rtve.es/webdocs/2014/12/la-realidad-virtual-como-afectar%C3%A1-a-los-webdocs-parte-5.html

[7] D. Circelli. "Historic Flight Simulator Offers Glimpse into Embry-Riddle's Past". Embry-Riddle Aeuronautical University. Disponible en: Historic Flight Simulator Offers Glimpse into Embry-Riddle's Past | Embry-Riddle Aeronautical University - Newsroom (erau.edu)

[8] C. Trilnick. "Sensorama". Proyecto Idis. Disponible en: Sensorama | IDIS (proyectoidis.org)

[9] R. Von Díaz "La Fundación BBVA premia a Ivan Sutherland, el padre de la realidad virtual". ElMundo, Ciencia y Salud. Disponible en: La Fundación BBVA premia a Ivan Sutherland, el padre de la realidad virtual | Ciencia (elmundo.es)

[10] T.A. Furness. The Super Cockpit and it's human factors challenges. Armstrong Aerospace Medical Research Laboratory Wright-Patterson Air Force Base, Ohio: September 1, 1986.

[11] E.B.Glenn. Chapter 3: Diverse Challenges Explored with Unified Spirit. *Atmosphere of freedom: sixty years at the NASA.* NASA SP 2000; 4314. pp.99-111. ISBN 0-9645537-2-4.

[12] S. Llamas. "Will the Oculus Quest be the answer to VR's prayers?". SuperData. Disponible en: https://www.superdataresearch.com/blog/will-the-oculus-quest-be-the-answer-to-vrs-prayers. [consulta en: 15 de noviembre 2020]

[13] P. Milgram, F. Kishino. "A Taxonomy of Mixed Reality Visual Displays". IEICE Transactions on Information Systems. E77-D, (12). USA 1994. Disponible en: http://vered.rose.utoronto.ca/people/paul_dir/IEICE94/ieice.htm (gmu.edu)

[14] F. Richter. "The diverse Potencial of VR & AR Applications". Statista. Disponible en: • Chart: The Diverse Potential of VR & AR Applications | Statista. [consulta en: 16 de noviembre de 2020]

[15] S. Agámez Luengas, M. Aldana Bolaño, V. Barreto Arcos, A. Santana Goenaga, y C. V. Caballero-Uribe. *Aplicación de nuevas tecnologías de la información en la enseñanza de la medicina*. Salud Uninorte [en linea]. 2009, 25(1), 150-171[fecha de consulta 8 de noviembre de 2020]. ISSN: 0120-5552. Disponible en: https://www.redalyc.org/articulo.oa?id=81711840013

[16] D. Malyasov. "Virtual Reality goes to work, helping train U.S. Army Soldiers". En: Defence-blog [en línea]. Disponible en: https://defence-blog.com/news/army/virtual-reality-goes-to-work-helping-train-u-s-army-soldiers.html (fecha de consulta: 8 de noviembre).

[17] G. Vera Ocete. *La realidad virtual y sus posibilidades didácticas*. Etic@net. Granada, España. Año II Número 2.ISSN:1695-324X. Disponible en: file:///C:/Users/anton/Downloads/Dialnet-LaRealidadVirtualYSusPosibilidadesDidacticas-6871642.pdf

[18] Cdra. y Lic. Roxana A. Pallare. "Realidad virtual e impresión 3D hidráulica en la industria del petróleo y del gas". *Petrotecnia.Revista oficial del Instituto Argentino del Petróleo y del Gas (IAPG)* [en línea]. pp 81-82, abril 2016. Dsiponible en: (http://www.petrotecnia.com.ar/abril16/Sin_Publicidad/Realidad.pdf). [consulta en: 10 de noviembre de 2020].

[19] E. Tellez. "Absolutamente real: la realidad virtual y aumentada abre nuevas vías en el sistema de producción de BMW Group". En: Press.bmwgroup. Disponible en: https://www.press.bmwgroup.com/mexico/article/detail/T0294390ES/absolutamente-real:-la-realidad-virtual-y-aumentada-abre-nuevas-v%C3%ADas-en-el-sistema-de-producci%C3%B3n-de-bmw-group?language=es

[20] L. Dupin. “VR is ready for consumers, are you ready for it?”. Awwwards. Disponible en: VR is Ready for Consumers, are You Ready for It? (awwwards.com). [consulta en: 16 de noviembre de 2020].

[21] “Oculus Rift. ¿Son las mejores gafas de realidad virtual para PC?”. GafasOculus. Disponible en: Oculus Rift de realidad Virtual - Características (gafasoculus.com). [consulta en: 15 de noviembre de 2020].

[22] J. Penalva “Los requisitos mínimos de un PC listo para VR”. Xataka. https://www.xataka.com/especiales/que-ordenador-comprar-si-quiero-usar-unas-gafas-de-realidad-virtual-guia-de-compras-con-pcs-compatibles

[23] “Historia de CMZ”. Disponible en: Historia | CMZ [consulta en: 11 de noviembre 2020]

[24] J. López Rodriguez. *“Clasificación de los procesos de mecanizado*”, en *Fundamentos de Procesos Convencionales de Fabricación Mecánica*. [en línea] Primera edición 2017, Cartagena. pp 12-52. ISBN: 978-84-16325-9. Disponible en: https://repositorio.upct.es/xmlui/bitstream/handle/10317/6068/isbn9788416325559.pdf?sequence=1&isAllowed=y. [consulta en: 12 de noviembre de 2020]

[25] Ivanov, A. “Documentation, Unity scripting languages and you”. Blogs.Unity3d. Disponible en: https://blogs.unity3d.com/es/2014/09/03/documentation-unity-scripting-languages-and-you/ [consulta en: 20 de noviembre de 2020 ]

[26] J. Horvath, E. Cosky y M. Franklin. “Design, Develop, and Deploy for VR”. Unity. Design, Develop, and Deploy for VR - Unity Learn. [consulta en: 4 de octubre de 2020].